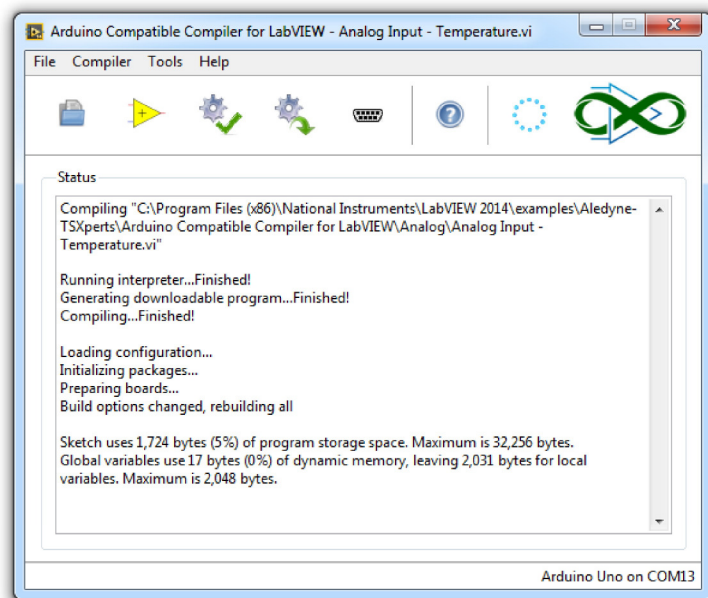


## Arduino Compatible Compiler for LabVIEW - Version 1.0.0.19

The Arduino Compatible Compiler for LabVIEW is a compiler that will take a LabVIEW Virtual Instrument (VI) and compile it for the Arduino™ programming language and will download the code to an Arduino™ board. The downloaded code will execute embedded and standalone on the Arduino target. The compiler allows a subset of VI features to be used. Please refer to the installed palette and help manual to see a list of supported features and APIs. Please be aware of memory constraints. The Arduino targets have very limited memory so care should be taken to limit or restrict usage of large and dynamic datatypes like strings and arrays, especially on smaller targets like the Uno.

You must first download and install the Arduino™ IDE version 1.5.7 or higher from <http://arduino.cc/en/main/software> to use this product.



Technical support requests to [support@tsxperts.com](mailto:support@tsxperts.com)

Arduino™ is a trademark of the Arduino Organization.

LabVIEW™ is a trademark of National Instruments Corporation.

©2015 TSXperts/Aledyne. All rights reserved

## Quick Start Guide

This guide is intended to provide detailed information that describes how the Arduino Compatible Compiler for LabVIEW works.

Understanding these details is fundamental to successfully create LabVIEW VIs which can be compiled and downloaded to an Arduino compatible target. It includes information about Arduino Compatible Compiler for LabVIEW programming concepts, step-by-step instructions for using the Arduino Compatible Compiler for LabVIEW, reference information about the Arduino Compatible Compiler for LabVIEW available APIs and references to the shipping examples.

This guide is organized in three main sections, [Licensing](#), [Getting Started](#) and [Important Considerations Before Start Creating Arduino VIs](#). It is strongly recommended the complete understanding of the contents of this guide, prior to the utilization of the compiler by a developer.

©2015 TSXperts/Aledyne. All rights reserved

## Getting Started with the Arduino Compatible Compiler for LabVIEW

Arduino Compatible Compiler for LabVIEW is a product based on LabVIEW (Laboratory Virtual Instrument Engineering Workbench) by National Instruments. LabVIEW is a graphical programming language that uses icons instead of lines of text to create applications. In contrast to text-based programming languages that use instructions to determine the order of program execution, LabVIEW uses dataflow programming. In data flow programming, the flow of data through the nodes on the block diagram determines the execution order of the VIs and functions. VIs, or virtual instruments, are LabVIEW programs that imitate physical instruments. This document assumes basic familiarity with LabVIEW. For more information about LabVIEW, visit [www.ni.com/labVIEW](http://www.ni.com/labVIEW).

The Arduino Compatible Compiler for LabVIEW is a true compiler, in the sense that a LabVIEW VI will be compiled into the Arduino compatible programming language, downloaded to the Arduino target and will execute embedded in the target. It is important to highlight a basic difference between the actual compilation of a LabVIEW VI into an Arduino target and a simple communication between an Arduino sketch running in the target and a LabVIEW VI, as provided by other existing toolkits. Any Arduino compatible hardware can be a target for the Arduino Compatible Compiler for LabVIEW.

Arduino is a tool for making computers that can sense and control more of the physical world than your desktop computer. It's an open-source physical computing platform based on a simple microcontroller board, and a development environment for writing software for the board. Arduino can be used to develop interactive objects, taking inputs from a variety of switches or sensors, and controlling a variety of lights, motors, and other physical outputs. Arduino projects can be stand-alone, or they can communicate with software running on your

computer (e.g. Flash, Processing, MaxMSP). This document assumes basic familiarity with the Arduino hardware. More about Arduino can be found at [www.arduino.cc/en/Guide/HomePage](http://www.arduino.cc/en/Guide/HomePage).

### What you Need to Get Started

The first step that needs to be taken prior to using the Arduino Compatible Compiler for LabVIEW is the installation of the Arduino IDE. The Arduino Compatible Compiler for LabVIEW works with the Arduino IDE version 1.5.7 and above. Download the latest Arduino IDE from [www.arduino.cc/en/Main/Software](http://www.arduino.cc/en/Main/Software). Another important point is that the Arduino IDE needs to be installed to its default location. If the user changes the directory the Arduino IDE is installed to during the installation process, the Arduino Compatible Compiler for LabVIEW will not be able to find it and an error will result when launching the Compiler.

At the time of this writing, the compiler was fully tested for code compilation and download to the following Arduino boards:

- Arduino Yun
- Arduino Uno
- Arduino Due
- Arduino Mega
- Arduino Leonardo
- Arduino Nano

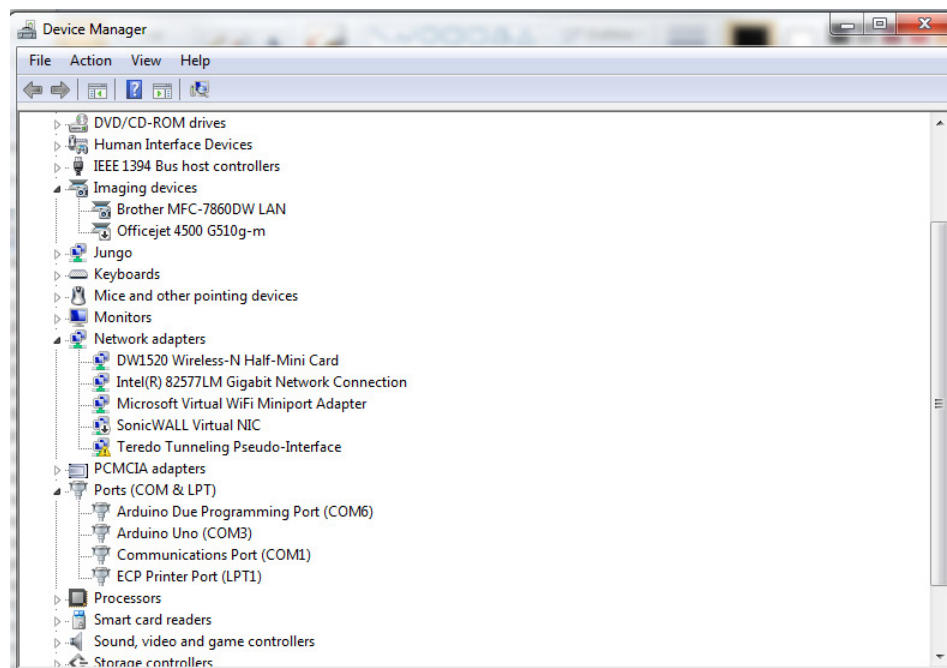
The following targets were tested for code compilation, but not for code download. Though there is a reasonable assumption these targets will behave similarly to the ones listed above and would therefore work with the compiler, they are not officially supported by TSXperts/Aledyne.

- Arduino Mega 2560
- Arduino Mega ADK
- Arduino Diecimila
- Arduino Micro
- Arduino Esplora
- Arduino Mini
- Arduino Ethernet
- Arduino Fio
- Arduino BT
- LilyPad Arduino
- Arduino Pro
- Arduino Pro Mini
- Arduino NG
- Arduino Robot Control
- Arduino Robot Motor

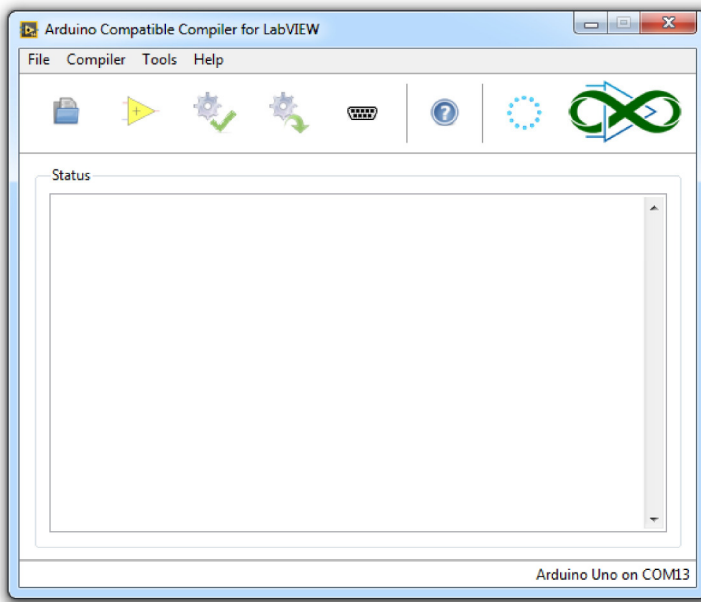
You will need one of the boards listed above and a USB cable connected to the PC running this compiler.

### Configuring the Arduino Compatible Target

Make sure your Arduino compatible target is connected to your PC and that the appropriate driver for it has been successfully installed. To verify this, navigate to your PC Device Manager and expand the Ports section. Under Ports, you should be able to see your connected Arduino, as shown by the following illustration.

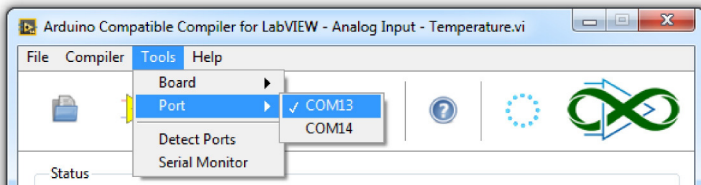


Now that you made sure your Arduino target is properly installed in your PC, launch the main compiler screen. To do that, open LabVIEW, and under its Tools menu, select "Arduino Compatible Compiler for LabVIEW". The following screen will be launched.

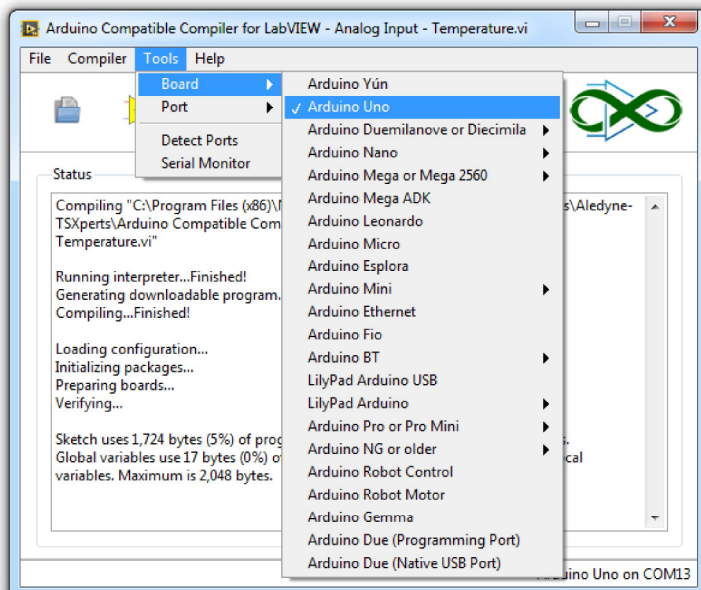


To configure a target for your VIs to be downloaded to, follow the steps below:

- 1) In the main compiler screen, select Tools from the main menu.
- 2) If a new Arduino target is connected to the computer while the compiler screen is open, you will need to refresh the connected ports. To execute this operation, select "Detect Ports" under the Tools menu. This option will refresh the list of serial ports available in your PC's operating system and make them available for selection from within the compiler screen. If all boards were connected to the computer prior to opening the compiler screen, you can skip this step
- 3) The last step to finish configuring your compiler is to select the COM port your target is connected to. To do that, select Port option under the Tools menu.



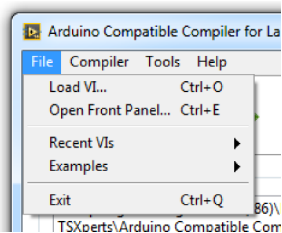
- 4) Select the board you wish to target from the list of supported boards, by clicking at the Board item under the Tools menu. Once the steps above have been performed, you should see the target configuration on the lower right hand corner of the main compiler window.



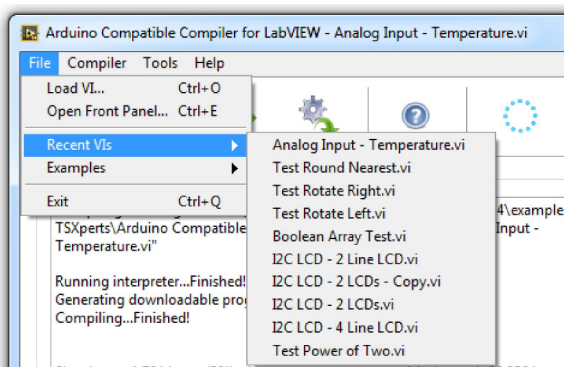
At this point, your compiler is ready to download compiled VIs to the configured Arduino compatible target.

### Compiling and Downloading a LabVIEW VI to your Arduino Compatible Target

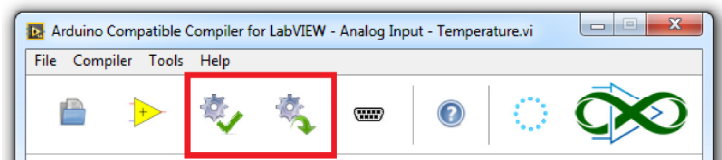
Once you have completed development of a LabVIEW VI that will be downloaded to the Arduino compatible target (for more information on how to successfully create LabVIEW VIs to be compiled to an Arduino compatible target, refer to section named [Important Considerations Before Start Creating Arduino VIs](#)), navigate to the File menu of the compiler and you will have the options illustrated by the following figure.



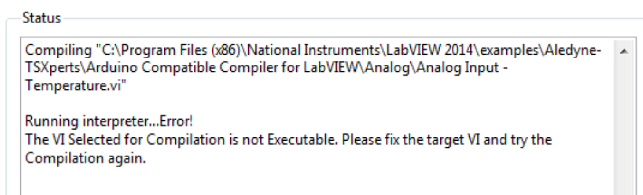
The Load VI option will point the compiler to the LabVIEW VI you wish to run embedded in the Arduino compatible target. You also have the convenience of opening the loaded VI for inspection if desired. For that option, select Open Front Panel under the File menu. You can also review a list of the recently loaded VIs for compilation under Recent VIs as illustrated below.



Once the VI has been loaded, you have the option to Compile (Verify) Only or Compile and Download the selected VI to the configured target, via the two buttons above the Status indicator.



The Compile Only option is the best way to verify if the loaded LabVIEW VI successfully compiles to the selected target. The Status indicator provides information about the result of the compilation that can be used for initial debugging in case the compilation fails. The following figure illustrates the example of a VI that was not in a LabVIEW runnable state when compilation was attempted. Other reported errors can be used to debug other compilation problems.

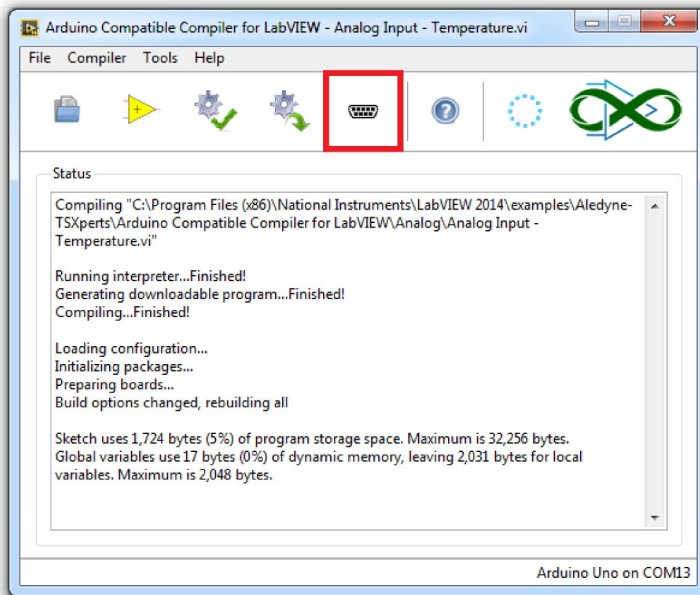


Once you have tested the VI compiles, you are ready to download it to the configured target. To do that, simply press the Compile and Download button and the LabVIEW VI will be downloaded to the target and will start execution.

### Serial Monitor

The Arduino Compatible Compiler for LabVIEW main screen includes a Serial Monitor application. The Serial Monitor application is launched by pressing the Serial Monitor button as displayed below.

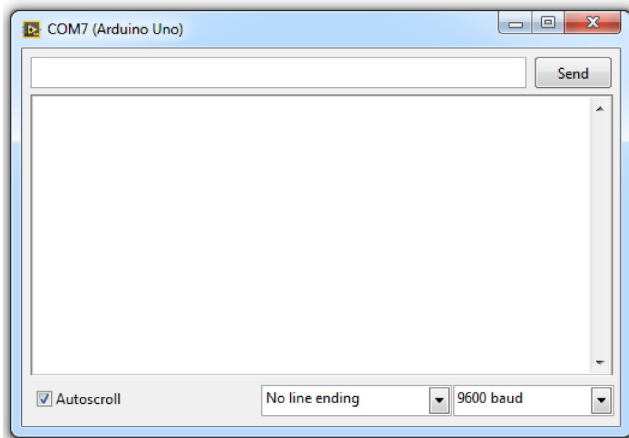




When pressed, the following screen is opened. The Serial Monitor will be initialized with the same serial port as configured in the main screen; in the case of the two figures shown, COM7. The default parameters will be set to "No line ending" and 9600 baud rate. If the "No line ending" parameter is selected, no extra characters are added to the end of the bytes to be sent via serial communication from the host PC. There are three other parameters the user can take advantage of: "New line", "Carriage return" and "Both LN and CR". The first one will add a new line, or 0x0A byte, to the end of string sent to the Arduino target. The second option will add a carriage return, or 0x0D, and the third option will add the two bytes, 0x0A 0x0D, to the end of the string.

The user has also the option to change the baud rate for serial communication. This setting should match the serial communication baud rate programmed in the running Arduino embedded VI. A 10-second timeout is configured for the serial connection to be established. If an error is generated, it means the PC could not establish the serial connection with the Arduino target with the configured parameters.

The Serial Monitor screen will display all incoming bytes that are read from the serial port and also allow the user to simultaneously send strings via serial communication. To send strings down from the PC to the Arduino target, type in the string in the control located to the left hand side of the "Send" button, then press the "Send" button.



## Finding Shipping Examples

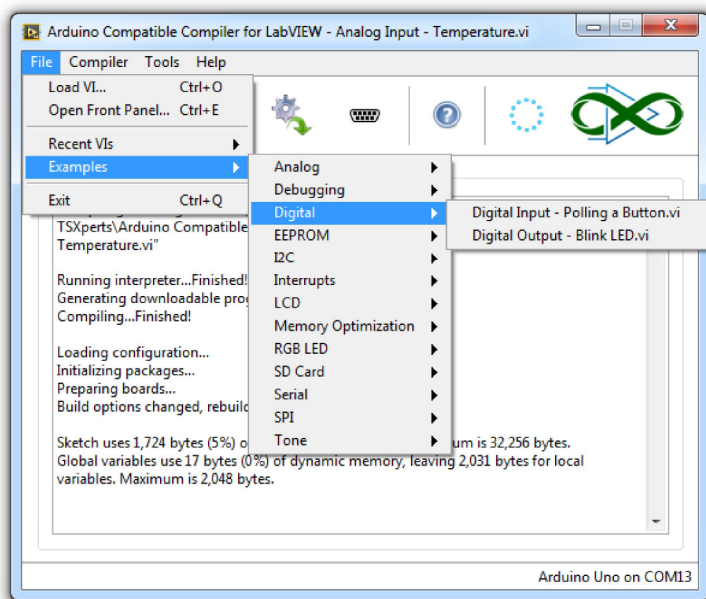
Arduino Compatible Compiler for LabVIEW ships with example VIs you can use and incorporate into VIs that you create. You can modify an example VI to fit an application, or you can copy and paste from one or more example VIs into a VI that you create.

Complete the following steps to browse all shipping Arduino Compatible Compiler for LabVIEW example VIs:

- 1) Launch National Instruments LabVIEW.
- 2) Select **Help>>Find Examples** to launch the NI Example Finder.
- 3) Navigate to the folder named Arduino Compatible Compiler for LabVIEW or search for the "Arduino" keyword.

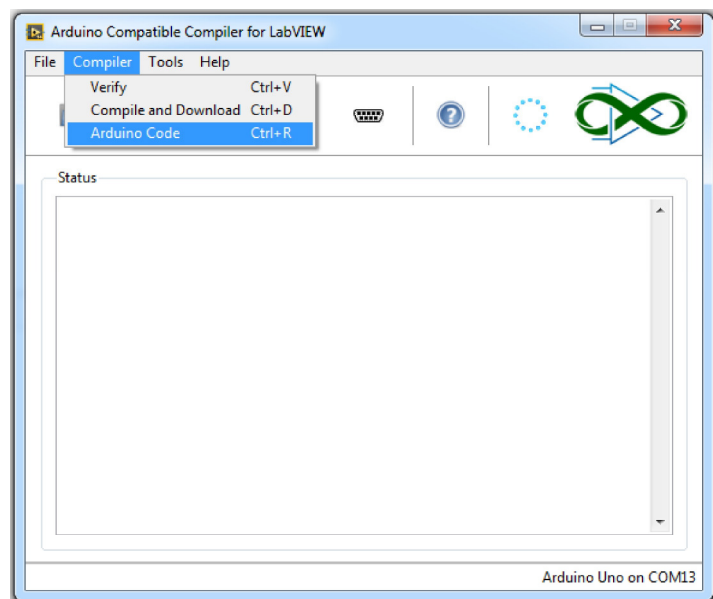
A comprehensive explanation of each example VI will be displayed in the right hand corner so you can select the one that best suit your immediate needs.

Another option to finding the shipping example VIs is via the main compiler screen as illustrated by the following figure.



### Porting an Arduino or Custom User Library to LabVIEW

The Arduino Compatible Compiler for LabVIEW includes the ability for users to port existing Arduino Libraries and create their own LabVIEW API VIs to bring a library's functionality into LabVIEW. This allows the community to take advantage of the existing ecosystem of Arduino shields and to create even more complex applications in LabVIEW for the Arduino platform. It is assumed at least a medium level of familiarity with the C language by the developer as example Arduino code that is part of the library will need to be read and understood as well as some lines of C-code will have to be written by the developer as part of the process. To view the converted Arduino Code for a user generated LabVIEW library, use the Arduino Code menu item shown below after loading a test VI. Refer to the [Porting an Arduino Library to LabVIEW](#) page for a complete guide to the process.



### Where to Go from Here

It is strongly suggested that the developer gets familiar with special details on how to best use the Arduino Compatible Compiler for LabVIEW. The section named [Important Considerations Before Creating Arduino VIs](#) contains information on what is supported, what is not supported, memory optimization techniques and other very important information for you to make the most out of your experience with the Arduino Compatible Compiler for LabVIEW.

©2015 TSXperts/Aledyne. All rights reserved

### Frequently Asked Questions

#### 1) Are there any characters that are not allowed in control and indicator names?

The Arduino IDE follows the general rules of the C programming language. Certain characters and names are not allowed and thus, should not be used in controls and/or indicators names. The Arduino Compatible Compiler for LabVIEW internally protects against the use of some of these characters and names, but not all. Care must be taken by the programmer to avoid the use of special characters and C

specific identifiers in variable names. For example, the following control names should be avoided even though they are allowed: String, enum, const, struct, int, word, boolean, char, byte, unsigned, signed, long. Care should be taken not to use similar names.

**2) LabVIEW for Windows uses "NaN", or Not a Number for calculations such as divide by 0 or a square root of a negative number. Does the Arduino Compatible Compiler for LabVIEW also support invalid calculations using NaN?**

No, the Arduino Compatible Compiler for LabVIEW will not return "NaN" in the event of an invalid computation. In fact, the embedded program will crash under the situation. Extra care needs to be taken by the program to protect from such calculations.

**3) My application doesn't perform the way I designed it. There are no compilation errors, but it is not consistent in the way it behaves and sometimes it just crashes. What is happening?**

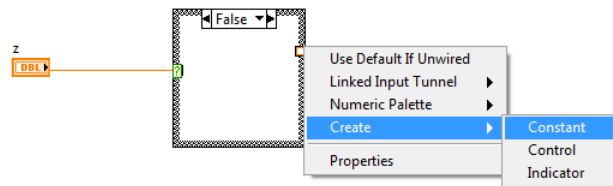
If your program is failing in an otherwise inexplicable fashion, the chances are good you have crashed the stack due to a SRAM shortage. This could be due to excessive use of dynamic memory or fragmentation caused by arrays or strings. On small memory targets, it is recommended not to use or to limit use of arrays and strings. For a complete account on memory considerations when programming your Arduino target with the Arduino Compatible Compiler for LabVIEW, please refer to the section entitled "Memory Management" in the user manual page named [Important Considerations Before Creating Arduino VIs](#).

**4) My LCD display is only showing 2 digits of precision when I write a floating point number directly to LiquidCrystal\_I2C write.vi. Why?**

The native LCD write API automatically truncates a floating point number to 2 significant digits. If you require more precision, use the Number to Fractional String primitive VI and wire the number of digits of precision needed to its corresponding input. Then wire the string output of this primitive VI to the LiquidCrystal\_I2C write polymorphic vi.

**5) Why do I get a compilation error when I right click at a unwired Case Structure tunnel and select the LabVIEW option "Use Default if Unwired"?**

The Arduino Compatible Compiler for LabVIEW currently does not support the "Use Default if Unwired" option for Case Structures. In order to avoid a compilation error, you will have to right click at the unwired tunnel and create a constant to be wired to the tunnel, as illustrated below.



**6) I have dropped a Compound Arithmetic node in my VI, right clicked on an input or output and selected the "Invert" option allowed by LabVIEW. Why is this not working when running the code?**

The current version of the Arduino Compatible Compiler for LabVIEW does not support Invert on the Compound Arithmetic node's inputs or outputs. You will need to use the LabVIEW "Not" primitive to invert the data before or after the compound arithmetic node.

**7) I have wired an enumerated constant with two elements to an Increment node. I would expect the output of the Increment to go back to the first element of the enumerated constant when incremented. But that is not happening. Why?**

In the current version of the Arduino Compatible Compiler for LabVIEW, enumerated data types do not wrap around back to the first element when its last element is incremented by one. Care must be taken to avoid the inclusion of that logic. This also applies for the decrement node.

**8) I wire a string constant "2223" to a Decimal String to Number primitive. I also wire a constant of "2" to the Offset input terminal. I expected to see "23" in the output of the primitive, but I am getting "2223" instead. Why?**

The current version of the Arduino Compatible Compiler for LabVIEW does not support the offset input terminal for any String to Number primitive VI. The entire string input will be converted to a number.

**9) I drop a Tick Count primitive in my VI, but I notice that everytime the Arduino board is power cycled, the Tick Count returns to 0. Why?**

The Arduino targets do not have an internal RTC (Real Time Clock) as part of its architecture. The RTC is the function of processors that maintains an internal tick count when power is lost or the microcontroller is restarted. Since the Arduino targets do not have that function, its internal counter will restart when the software is restarted.

**10) Why can't I use Clusters in my VI with the Arduino Compatible Compiler for LabVIEW?**

Clusters are indeed a great way to organize complicated data structures. However, since the Arduino targets are small microcontrollers in nature, part of the memory optimization implemented in the compiler was the suppression of Clusters due to its inherit overhead.

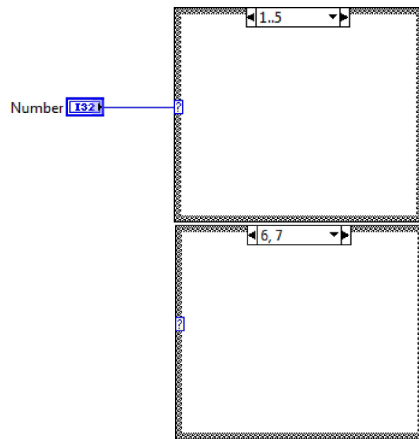
**11) Are 64-bit data types supported? What about fixed point numbers? Where can I find a list of all data types supported by the Arduino Compatible Compiler for LabVIEW?**

64-bit and fixed point numbers are not supported. For a full list of all data types supported by the Arduino Compatible Compiler for LabVIEW, refer to ???TBD

**12) Can I wire a string variable to the case selector terminal of a Case Structure?**

The short answer is yes, it is supported by the compiler. However, it is not recommended since each string case will take up a significant amount of memory to store the case selector label. It is recommended to use enumerated data types instead of string variables to wire to the case selector of a Case Structure.

**13) On Case Structures, can I specify a range and multiple items in a Case, as illustrated by the figure below?**



Yes, ranges and multiple items on Case Structures are supported by the Arduino Compatible Compiler for LabVIEW.

**14) Can I use property nodes for my VI controls and indicators?**

No, property nodes are not supported by the Arduino Compatible Compiler for LabVIEW.

**15) I am getting a long compilation error stating there is no match for 'operator=' when I use a subVI. Why?**

SubVI\_Array\_Test.ino:117:28: error: no match for 'operator=' (operand types are 'LVOneDimArray<short unsigned int>' and 'LVOneDimArray<long unsigned int>')  
SubVI\_Array\_Test.ino:117:28: note: candidate is:

You probably have a data type mismatch between the subVI input terminals and the variables that are wired to such terminals. The Arduino Compatible Compiler for LabVIEW currently does not support automatic coercion of datatypes wired to subVI input terminals and will generate a compilation error.

**16) Can I embed the Arduino Compatible Compiler in another VI I created?**

Yes, there is a [Compile.vi](#) that is included in the Compilation subpalette of the Arduino Compatible Compiler for LabVIEW palette that can be used to automate the compilation or compilation and download of a VI to an Arduino target. There is also a [command line interface](#).

**17) I have a password protected subVI named Test.vi as part of my top level VI. Why do I get a compilation error stating: "Arduino function "Test" not supported"?**

The Arduino Compatible Compiler for LabVIEW does NOT support password protected subVIs. If such subVI is used as part of a VI, you will get the compilation error you reported.

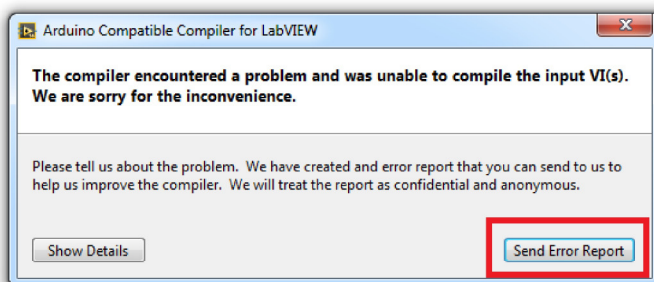
©2015 TSXperts/Aledyne. All rights reserved

**Important Considerations Before Creating Arduino VIs**

The Arduino™ hardware is composed of a single small microcontroller and some basic components, to which your VI will get compiled and downloaded to. As with any embedded target, some important considerations about memory optimization need to be made in order for your target to run more complex programs.

The LabVIEW programming language was created with the regular personal computer in mind as the target. Moreover, LabVIEW has graphical user interface capabilities that, obviously, are not applicable to a headless embedded target.

For convenience, the Arduino Compatible Compiler for LabVIEW installs its own function palette into the LabVIEW functions palettes. It is highly encouraged that the user work from within this palette when creating VIs to be compiled to an Arduino target. VIs from other palettes are not supported by the compiler and their use will cause a compilation error as follows:

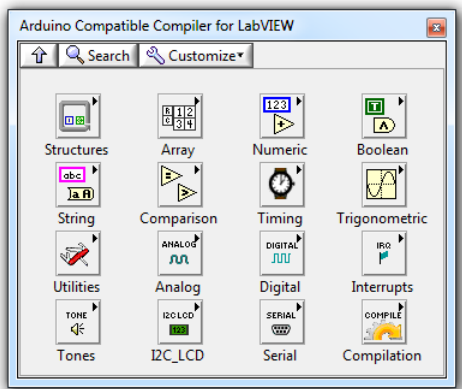


If the above exception occurs the "Send Error Report" button can be used to send a report of the error to the developers of the Arduino Compatible Compiler for LabVIEW. This information will be very helpful in improving the product in future releases.

To access the Arduino Compatible Compiler for LabVIEW palette, take the following steps:

- 1) Open a new LabVIEW VI
- 2) Navigate to the VI block diagram

- 3) Right click at the block diagram and navigate to Addons
- 4) You will see the Arduino Compatible Compiler sub-palette
- 5) Pin the Arduino Compatible Compiler sub-palette down so it constantly shows while you create your VI code



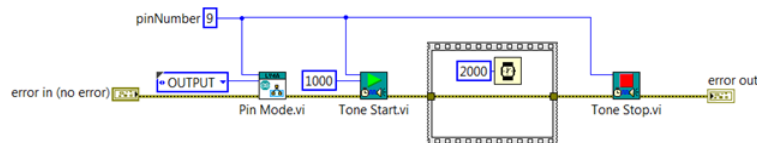
This section will present some fundamental details that need to be fully understood by the developer, prior to creating Arduino™ compatible VIs. Make sure to read the entire document as it is an important preparation step to successfully create Arduino™ compatible LabVIEW VIs.

### Clusters

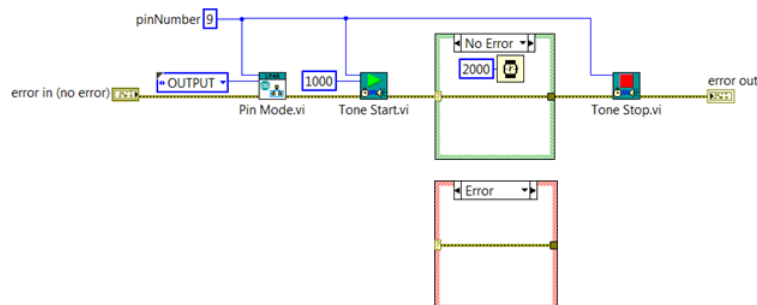
Though clusters are a great way to organize data in LabVIEW, they do create extra overhead to the generated code. Therefore, in the interest of minimizing unnecessary overhead and getting the most out of small Arduino targets, clusters are not supported by the Arduino Compatible Compiler for LabVIEW. The compiler will throw an error if a cluster is used in an Arduino™ VI. However, there is one small exception to this rule, the native LabVIEW Error Cluster.

Data flow programming is the basis of the LabVIEW programming language. In summary, the order which a LabVIEW VI executes is determined by the way that data gets transferred across wires on the block diagram. That is called data flow programming. For more information on data flow programming, visit <http://www.ni.com/video/1875/en/>.

A good practice in creating a LabVIEW VI is to always make sure the programmer is in control of the execution order of the VI. Such practice helps in avoiding race conditions and some other very difficult to debug issues with the code. The utilization of the Error Cluster as a mechanism to control data flow is good practice to ensure data flow is being properly utilized. The following VI utilizes this concept to ensure a delay of 2000 milliseconds will be executed after Tone Start has completed execution and before Tone Stop starts executing.

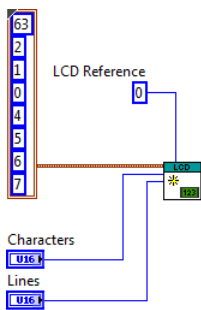


As such, LabVIEW Compatible Compiler for Arduino will not generate a compilation error due to the utilization of the native LabVIEW error cluster. The VI shown above will be compiled in a way that the 2000 milliseconds delay will execute after the Tone Start API function finishes its execution and Tone Stop API function starts its execution, following data flow programming. However, it is important to highlight the fact that the information carried by the error cluster itself is not meaningful to the compiler. This means code like the snippet below will not generate the expected result, and a compilation error will be generated.



### Typedefs

Typedefs are also a great way to organize data structures. However, for the same reason as clusters are not supported, typedefs are not supported by the compiler either. The only exception to this rule is for information passed into Arduino Compatible Compiler specific APIs that take a typedef as an input, as shown in the next figure. This code snippet will compile with no issues.



## Memory Management

As mentioned previously, the Arduino™ target requires stringent memory management in order to execute complex applications. Therefore, care must be taken by the programmer to maximize the potential of the target. This section explains how the compiler handles memory management and optimization, and outlines good practices that should be followed by the programmer to achieve better memory management. Most of the examples presented in this section illustrate the use of arrays; however, note that the same optimization algorithms that are applied to arrays are also applied to string data types by the compiler. **It is extremely important to note that, although the compiler includes a level of optimization algorithms, the use of arrays and strings are highly discouraged when targeting the Arduino Uno and similar targets, due to their very limited available RAM.**

One of the biggest differences between LabVIEW and an embedded programming language, like ANSI C, is that LabVIEW abstracts memory allocations from the programmer, whereas its text based counterparts leave that responsibility to the programmer. This certainly makes life much simpler for a LabVIEW programmer who can freely create variables and the LabVIEW compiler takes care of their corresponding memory allocations in the background. That, however, can cause issues if the programmer is creating a VI to be executed on an embedded target and doesn't pay attention to how memory is being allocated in the background.

It is certainly useful that the programmer understands how LabVIEW handles memory allocations. This document will present a very brief overview of some basic concepts. For a more complete account of how LabVIEW handles memory allocation, refer to [this](#) presentation. It will use array type variables in the explanation as it is the one with the most significance for our specific case and the easiest one to understand, though the same concepts can be applied to other data types as well.

In general, the Arduino™ memory is divided into four different areas:

- The code area, where the compiled program sits in memory (resided in Flash).
- The globals area, where global scalar variables are stored.
- The heap, where dynamically allocated variables are allocated from (arrays and strings).
- The stack, where parameters and local variables are allocated from when calling functions (SubVIs).

The global area, heap and a stack are stored in SRAM. SRAM is the most precious memory commodity on the Arduino™. Although SRAM shortages are probably the most common memory problems on the Arduino™, they are also the hardest to diagnose. If your program is failing in an otherwise inexplicable fashion, the chances are good you have crashed the stack due to a SRAM shortage.

The Arduino Compatible Compiler for LabVIEW handles allocating variables to each of these memory locations differently for each datatype.

For scalars, such as integers, floating-point scalars, and booleans that are not defined as a global or do not have a local variable reference, automatic-duration variables are used. Automatic-duration variables are allocated on the stack and come and go as subVIs are called and return. The advantage to this is that if a lot of wire nodes are used, it is helpful to modularize the code into smaller subVIs since nodes local to those VIs will be placed in memory on the stack, and will automatically be released when the subVI completes execution. The disadvantage is that there may not be enough memory available on the stack to support the required memory for a particular subVI call. This would not be known until run-time and may cause unexpected behavior or an unexpected exception of the executable code. Placing all code at the top-level VI will mean that all nodes (and associated variables) will be created on the stack local to the main VI and will exist for the entirety of the program execution. But the stack must be large enough to support all those memory allocations.

For scalar types that are defined as either global variables, or have an associated local variable reference, static-duration global variables will be used. Static-duration variables are allocated in global memory and persist for the lifetime of the program. The advantage is that it can be known at compile time if there is enough memory on the target to support the required memory. Global and static variables are the first things loaded into SRAM. They push the start of the heap upward toward the stack and they will occupy this space for all eternity.

For arrays and strings, dynamic memory allocation is used. This is required due to the native operation of LabVIEW in order to support the complex behavior of these datatypes. Dynamic memory allocation is the process of memory resizing as a given variable dynamically grows as the code executes. Dynamic memory is stored on the heap. In the case of arrays, for example, if new array elements are added to an array variable or node, the memory allocated to that array variable needs to grow in size. The downside to using dynamic memory allocation is that it cannot be known at compile time if there is enough RAM to support the required data and variable usage. If there is not enough RAM to allocate or resize a variable at run-time, it can cause the program to behave unexpectedly or even stop working completely. However, the advantage to dynamic memory allocation is that nice features and functions like concatenate strings, build array, and auto-indexing of LabVIEW structures can be supported.

Dynamic memory allocation also tends to be non-deterministic; the time taken to allocate memory may not be predictable and the memory pool may become fragmented, resulting in unexpected allocation failures. Memory fragmentation occurs when most of the memory is allocated in a large number of non-contiguous blocks, or chunks - leaving a good percentage of the total memory unallocated, but unusable for most typical scenarios. This results in out of memory exceptions, or allocation errors in which the Arduino Compatible Compiler for LabVIEW tries to detect in order to prevent a complete crash of the program. However, the result is unexpected data behavior.



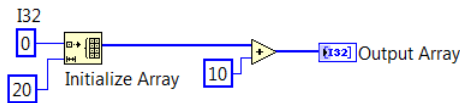
In summary, if the target does not have enough memory to allow all allocations, the program can produce unexpected results or can even crash and stop working without further warning. It is important to reduce usage of arrays and strings to avoid such problems.

## Memory Optimization

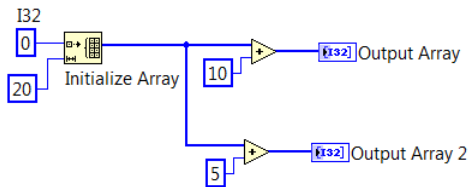
Another important concept is the idea of reusing declared variables as much as possible, as opposed to creating a new variable copy for each input/output variable of each called function. This concept is called inplaceness optimization.

Inplaceness optimization happens when an algorithm, or function, reuses one or more of its input variables by overwriting the output data that is generated by said algorithm or function to the same input variable memory space, as opposed to creating a separate variable to store the output data. Intuitively, one can see how this technique saves memory space in an executing program. For small targets, this becomes of special concern, especially for long sized arrays. A target can very quickly run out of memory resources in the event that no inplaceness optimization is applied, especially if large arrays are part of the program.

The Arduino Compatible Compiler for LabVIEW does implement inplaceness, following the same general rules that the LabVIEW compiler does for inplaceness optimization. Let's take an initial simple example to illustrate the concept.

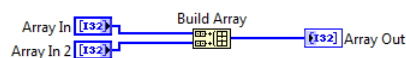


In the example above, the Initialize Function will create an array of 20 elements of type I32. The created array will serve as the input to the Add function, which will add 10 to each element of the input array. In this case, since the input array is of fixed-size, the LabVIEW compiler, and the Arduino Compatible Compiler for LabVIEW, will reuse the input array memory via inplaceness optimization to serve as the memory space for the **Output Array** variable. In this example, only one instance of an array variable is created and reused throughout the entire execution. Let us now evaluate a second example.



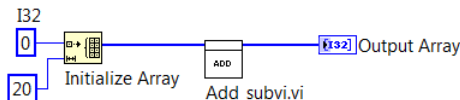
This example extends the code for the previous example by adding an extra Add function to be executed to the same array that was created by the Initialize Array function. In this case, it is not possible for the compiler to reuse the same memory space at the output of the two separate Add functions. Because the output of the initialize array function has a wire branch, the compiler must create two copies of data for the inputs to each add function.

As a rule of thumb, when an array variable, or in LabVIEW terminology - a wire, is branched to be consumed by more than one function, the compiler will not be able to do in-place optimization and an extra copy of the array will be created. This is an extremely important concept by embedded programmers. Furthermore, only functions that don't modify the size of the array as part of its operation are candidates for in-place optimization. On the examples presented above, the size of the input array to the Add function is maintained intact at the output of the function; only the array elements are operated on. Therefore, the Add function is a candidate for inplaceness. Let's take the case of another LabVIEW array function widely used; the Build Array.



On the example above, the Build Array with "concatenate inputs" enabled is used. The output array is a one dimensional array, as are the two input arrays. However, **Array Out** will have as many elements as the sum of the elements from **Array In** and **Array In 2**. Therefore, inplaceness optimization can't be applied here. Another in-place optimization rule to be kept in mind is, inplaceness can not happen if the function modifies the size of the array inputs.

Now, let's expand this concept onto a special node in a LabVIEW diagram, a subVI.



In the example presented above, a user created subVI named **Add\_subvi.vi** is part of the diagram. The subVI receives an array input, process it, and returns a single array output. Even though there is no branching of the input array to the subVI node, the Arduino Compatible Compiler for LabVIEW does not operate in-place for this array. In summary, even though subVIs are a great way to make the overall LabVIEW application more readable and modular, their utilization does generate memory overhead to the final compiled code that is downloaded to the Arduino target. This is an important tradeoff the programmer needs to consider in order to organize the overall application to have a good level of modularity and readability, but also enable execution without problems on small Arduino targets.

An alternative to passing arrays to subVI inputs is to use global variables. This is common practice in embedded programming; where a large array is read or modified in functions or subroutines as a global variable. In LabVIEW specific terminology, this would be the equivalent to creating a global variable array that is then accessed from within a subVI. Obviously, the programmer needs to be very careful with potential race conditions that can be introduced with the user of Global Variables. Refer to the shipping examples for an example of utilization of global variables to access an array from within a subVIs without passing a wire in or out of the SubVI.

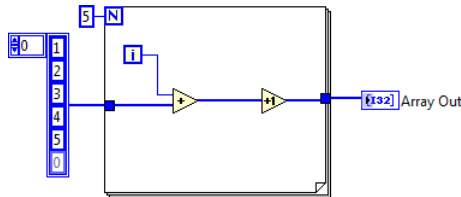
One last point that is important in regards to optimization of subVIs is to describe how the diagram *inside* the subVI is optimized. For the

sake of illustration, assume the **Add\_subvi.vi** in the example above implements the following diagram.



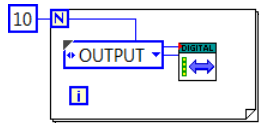
In this case, the original rules of inplaceness optimization are followed for the diagram inside a subVI. More specifically, in the diagram presented above, since the incoming array to both the Add and Increment nodes are not branched, inplaceness optimization will be applied and a single memory copy will be created to handle the array operations for the entire diagram. To summarize how subVIs are handled in regards to memory optimization, an extra memory copy is created for input variables, or in textual programming language terminology, a local variable is created on the stack (or on the heap for strings and arrays). However, from that point on, the inplaceness optimization rules are applied to the local variable. However, at the end of the routine, another copy of the variable must be made to handle the subVI output.

The next scenario to cover in regards to inplaceness optimization is the utilization of structures. A LabVIEW structure is herein being defined as any node that contains a diagram in itself, such as While Loops, For Loops, Case Structures and Flat Sequence Structures. Take the following example:

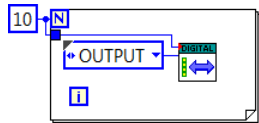


The exact same logic explained for the case of the subVI optimization is applied to the case of structures. In the example above, a memory copy is created to handle the input variable to the For Loop, the array constant, and another one to receive the output value from the For Loop. However, inside the For Loop, a single memory copy is used to handle all operations that modify the input array.

One other case to be considered is the wiring of the For Loop "N" terminal, inside the For Loop diagram. Consider the following scenario:



On this case, the compiler will create an extra variable to store the value of N, which will be passed to the input of the Digital API VI declaration. The above code can be replaced with the following code, and the functionality is maintained:

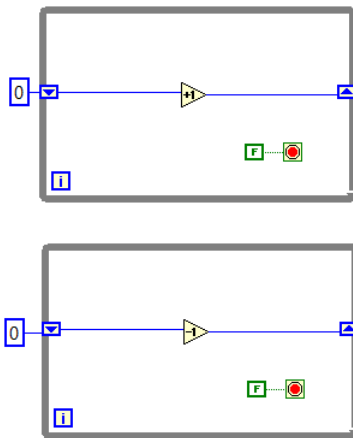


On the later case, the variable that stores the constant "10" is reused inside the For loop and passed in as an input to the Digital API VI declaration. This implementation avoids the creation of the extra variable to store the N value, optimizing memory utilization.

### Parallel Loops and Multi Threading

LabVIEW and its graphical programming paradigm are very conducive to parallel processing. In fact parallel programming is one of LabVIEW's strengths. The LabVIEW compiler even goes as far as splitting thread allocation amongst the multiple cores of a multi core CPU.

However, Arduino, as mentioned previously, is a non-Operating System microcontroller target. Parallel computing and multi-threading are best handled by CPUs running an operating system with a scheduler coordinating multiple threads. Microcontrollers are not the best targets for the execution of such architectures. Moreover, the Arduino Uno is a very small microcontroller based target. Since the Arduino Compatible Compiler for LabVIEW was created to support all Arduino targets released by the Arduino organization, including the Arduino Uno, the compiler does **not support parallelism** of loops. Let's illustrate the concept through an example.



The example above shows two independent parallel loops free running in an infinite loop. In regular LabVIEW, this code actually executes exactly as such; two parallel loops doing computation with no stop condition.

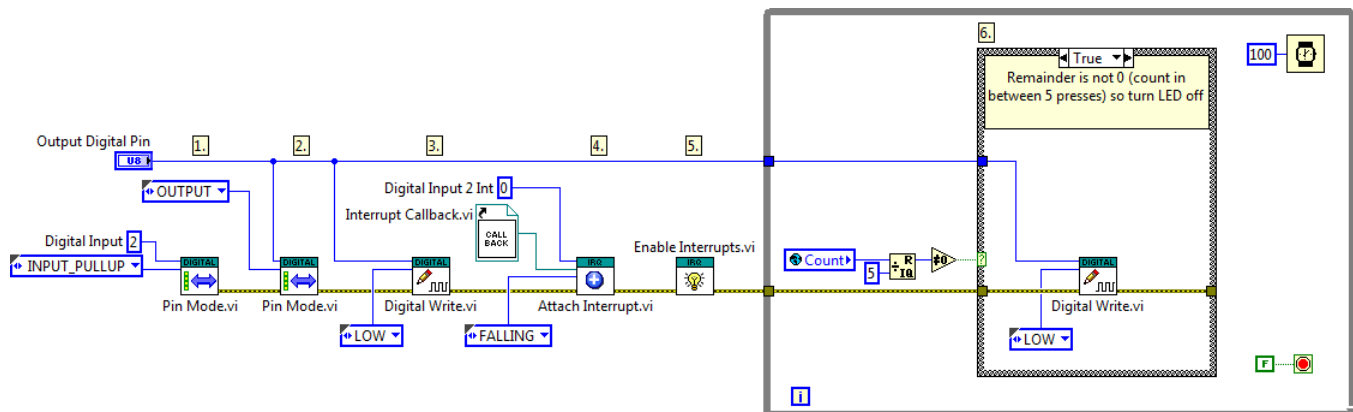
If the above VI is compiled using the Arduino Compatible Compiler for LabVIEW, since the compiler doesn't support multi-threading, the code will be compiled but execution will happen sequentially. Meaning, one loop will execute through completion, and following, the second loop will start its execution. In this case, since both loops are infinite loops, only one of the loops will execute and the second will never get executed.

Since there is no multi-threading in the target, the concept of thread priority does not apply to the Arduino Compatible Compiler for LabVIEW. Therefore, one cannot predict which of the two loops will be placed ahead of the other and will be executable. The best way to avoid situations like this is to use data flow to control the order of multiple loop execution if an application requires more than a single loop.

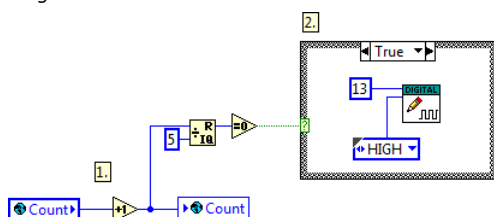
### Interrupts Programming and Handling

One of the most useful functions of an embedded target is its ability to generate interrupts and allow the programmer to create what is called interrupt handler functions. An interrupt is a signal to the microcontroller emitted by hardware indicating an event that needs immediate attention. An interrupt alerts the microcontroller to a high-priority condition requiring the interruption of the current code the microcontroller is executing. The microcontroller responds by suspending its current activities, saving its state, and executing a function called an interrupt handler (or an interrupt service routine, ISR) to deal with the event. This interruption is temporary, and after the interrupt handler finishes, the processor resumes normal activities. Refer to your specific Arduino target documentation for specific information about how many interrupts it allows and in what pins.

The Arduino Compatible Compiler for LabVIEW allows the programming of interrupts as well as the creation of an interrupt handling routine for the generated interrupt. The following diagram is an actual shipping example that can be used as a starting point for the creation of interrupt driven Arduino VIs.



The diagram above shows the specific Interrupt API VIs that ship with the compiler and allows interrupts to be configured in the Arduino target. It also shows how the interrupt handling routine is configured. The use of a Static VI Reference as an input to the Attach Interrupt API VI allows the user to tell the compiler which subVI will be the actual interrupt handling function. In this case, it sets the Interrupt Callback.vi to be the subVI that will function as the interrupt handling routine. The following figure illustrates the Interrupt Callback diagram.



It is important to note that global variables are used to perform data exchange between the main VI and the interrupt handling subVI.

### Exception Cases

Though great effort was applied to make sure the supported functionality of the Arduino Compatible Compiler for LabVIEW will allow the same level of flexibility and configurability that LabVIEW itself offers, the compiler does present a few differences that are important to be highlighted.

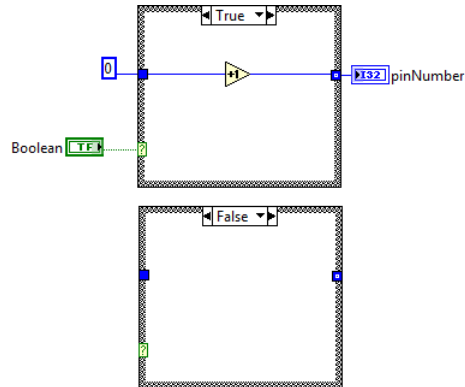
#### Forbidden Characters for Controls and Indicators

The forward and back slash; "/" and "\" respectively, as well as the following characters ":", ";", "#" and "&" can NOT be part of any controls or indicators names. They are special characters that are used by the Arduino Compatible Compiler for LabVIEW. Therefore; the following control names; "X/Y", "X\Y", "X:Y", "X;Y", "X#Y" and "X&Y" are illegal and will generate a compilation error.

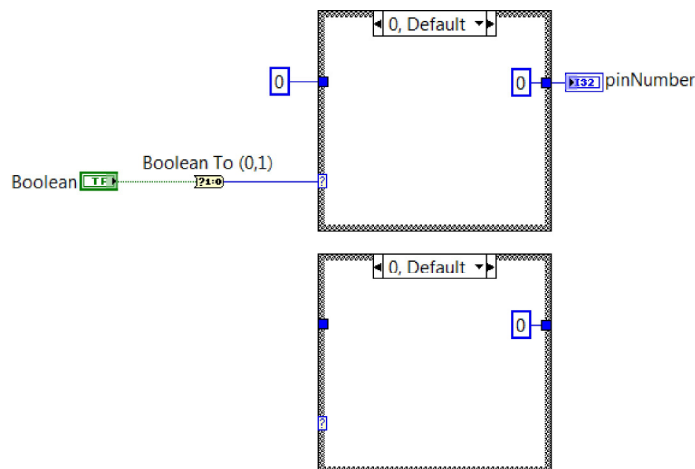
#### Boolean Case Statement on non-English Versions of LabVIEW

One of the differences between how LabVIEW and the Arduino Compatible Compiler for LabVIEW work is related to boolean case statements.

LabVIEW English version will show the following implementation of a boolean case statement.



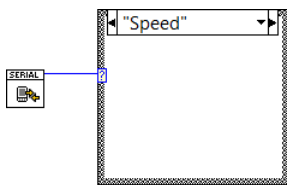
Non-English versions LabVIEW will, however, translate to the corresponding localized language the True/False words that define the frame names for a boolean case statement. This causes the LabVIEW Compatible Compiler for LabVIEW to throw a compilation error. There is however a workaround for this issue, the implementation above would need to be changed to the following in order for it to compile without errors.



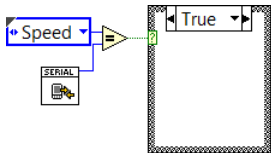
Note how a Boolean To (0,1) function, from the Boolean palette is being used to convert the boolean value that would be wired to the case statement into an integer value. For this situation, the "1" frame will implement the code that was originally inside the True frame and the "0" frame will implement the code that was inside the False frame, in the boolean case statement scenario. The overall functionality is identical; however an extra step is required in the case of the Arduino Compatible Compiler for LabVIEW.

#### Enum Case Statement Wired from SubVI

Due to the way the compiler was designed, case statements only support Enum types wired to their case selectors if the Enum is part of the same diagram; i.e. doesn't come from a subVI. The following code snippet, showing an Enum type coming from a SubVI, would generate a compilation error.

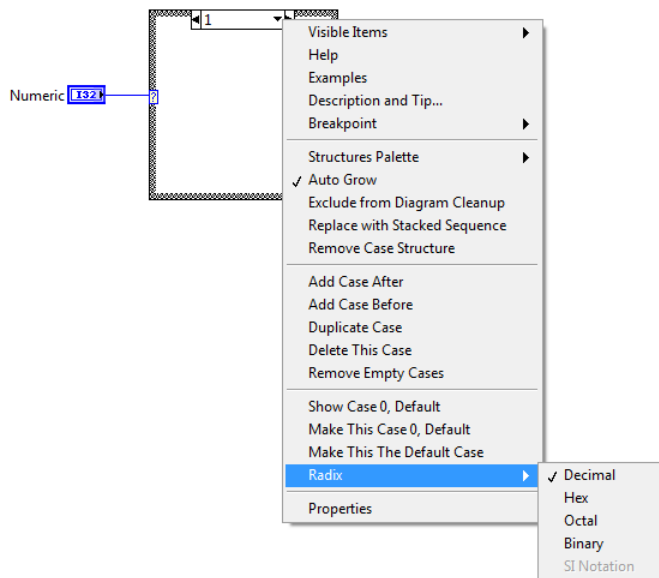


The workaround for this is to perform a comparison external to the subVI that would turn the Case Statement into a Boolean type; as illustrated below.



#### Case Structure Radix Setting:

LabVIEW allows the programmer to right click at Case Structure and change the display Radix.



This allows the programmer to switch the Case Statement over to Hexadecimal, Octal or Binary numeric data types, in addition to the Decimal data type. The Arduino Compatible Compiler for LabVIEW does NOT support any Case Statement Radix other than the Decimal data type. Especial care should be taken by the developer to avoid this optional selection as, depending on the values entered by the user, the compilation may actually complete without errors. However, the functionality of the deployed VI may differ from the intended functionality by the programmer.

#### Password Protected SubVIs

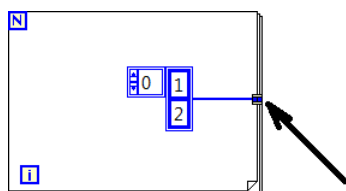
The Arduino Compatible Compiler for LabVIEW does NOT support password protected subVIs. If such subVI is used as part of a VI, a compilation error is generated.

#### String Arrays

The Arduino Compatible Compiler for LabVIEW does NOT support commas as part of string array elements. If your VI includes either a string array constant or a string array control with default values, you need to make sure the string array elements do not include a comma as part of their values. If a comma is used on these scenarios, the Arduino Compatible Compiler for LabVIEW will not generate a compilation error, however the compiled output code will be erroneous and the code functionality will be at risk.

#### Concatenate Output Tunnel

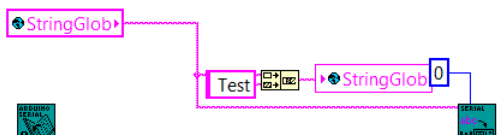
LabVIEW offers an option for the user to set an output tunnel of a loop to concatenate its elements, as illustrated in the figure below. The Arduino Compatible Compiler for LabVIEW currently does NOT support this feature.



### Global Variable Utilization

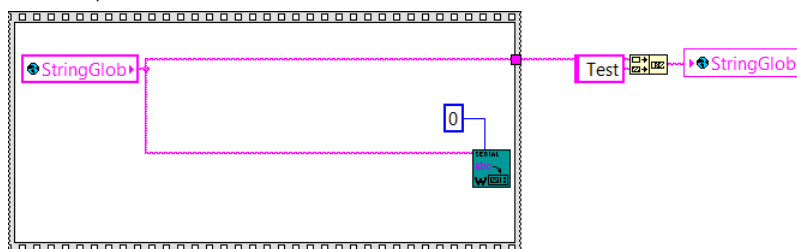
As mentioned previously, the use of global variables is a great way to work around the limitations of the small hardware footprint of the Arduino targets. However, since global variables break the concept of data flow, there are a couple of exceptions on its utilization that the user needs to pay attention to. These exceptions are documented below.

The first exception to be highlighted is displayed in the following code snippet.

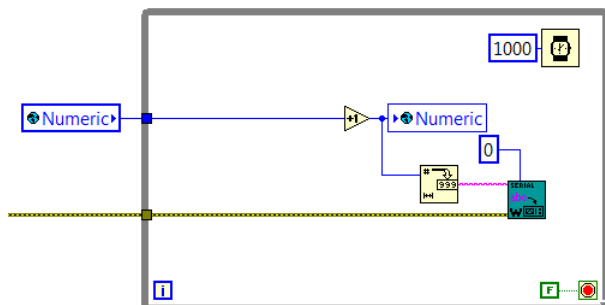


As one can see in the code segment above, the global variable is being used in the input and output of a node and is also being wired, in parallel, as the input of a subVI. The way LabVIEW handles this is such that the global variable is passed to the subVI and only then updated with the output of the concatenate function. However, the way the Arduino Compatible Compiler for LabVIEW was designed, since there is no data flow dependency between the concatenate function and the subVI, there is no way to, a priori, if the concatenate function will run first or the subVI will run first. If the concatenate runs first, the global variable will be updated with the output string of the concatenation operation. Then, when the subVI executes, the global variable value that will be used as the input of the subVI will not be the original value prior to the concatenation operation, but the value after concatenation. This generates a race condition on the code.

The workaround to this issue is similar to the workaround to avoid global variable race conditions in general; which is to make sure the order of utilization and update of the global variable is forced through data flow. If that is not possible, as in the example shown above, the utilization of a sequence structure is advised, as it forces the execution to happen in the order that it should to avoid the race condition, as illustrated below.



The second exception to be documented happens in the event of a global variable read outside a structure (for loop, while, loop, case statement, etc) and subsequent update to the same global variable inside the structure, as illustrated by the figure below.



The way LabVIEW handles this scenario is such that the output of the increment function will never change value, as a new variable is created to hold the input tunnel value, which is not connected to the global variable. Since the Arduino Compatible Compiler for LabVIEW implements very stringent optimization algorithms to try and reduce the utilization of the Arduino target internal memory, on this situation, the input tunnel variable is actually the global variable itself, as a copy of the global variable is not made to hold the input tunnel value. The issue that this optimization will create on this particular case is that the output of the increment function will continuously increment the value, as its output updates the value of the global variable and its input receives the same global variable in every iteration. On this case, the workaround is very simple, just move the global variable to the inside of the loop. In general though, care should be taken when a global variable is being passed in as an input to a structure, so this situation can be avoided.

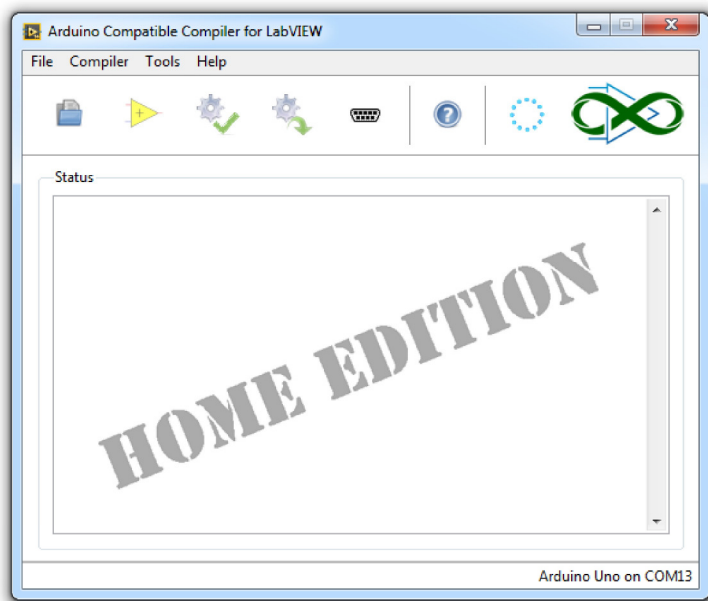
### Other Non Supported Features:

There are some non-supported data types and features that are specific to supported nodes. Please refer to the documentation for the supported functions for more information on each one.



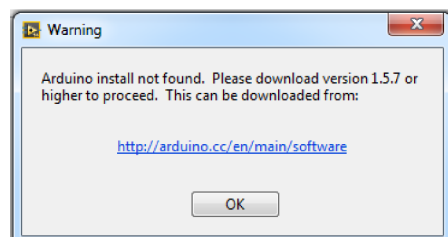
### Arduino Compatible Compiler for LabVIEW Licensing Options

The Arduino Compatible Compiler for LabVIEW is offered in two versions; Home and Standard. There are no functional differences between the two versions of the tool. All supported LabVIEW primitives and Arduino™ API VIs are equally available with both versions. For a complete list of all supported features of the product, please refer to the product User Manual. However, the Home version is intended for use with the Student and Home editions of LabVIEW 2014 or later and is only licensed for use on personal, home, or educational projects. The Standard version is licensed for use in the professional market and works with any edition of LabVIEW 2014 or later. Another minor difference is that the Home version will show a watermark, indicating the license is only allowed for Home or Student use, as illustrated below.



The Home version of the compiler is tailored to support the makers movement, home projects and academic use, being offered at a considerable discount to the community. The Standard version is targeted for the professional setting, for applications expected to run with LabVIEW Base or Professional editions. Both version options offer a 7-day evaluation period, during which the user is free to run the full-featured product without purchasing a license. Once the evaluation period is over, the user will be prompted to enter a purchased license key to continue use of the tool.

The Arduino Compatible Compiler for LabVIEW also relies on the installation of the Arduino IDE version 1.5.7 or later, which must be installed on the same computer the compiler is installed. The user needs to make sure the computer has a compatible version of the Arduino IDE according to the above. The Arduino IDE can be downloaded from [this link](http://arduino.cc/en/main/software) prior to attempting to run the Arduino Compatible Compiler for LabVIEW. If the Arduino IDE is not installed, the following error will be displayed:



#### Porting an Arduino Library to the Arduino Compatible Compiler for LabVIEW

The Arduino Compatible Compiler for LabVIEW ships with LabVIEW API VIs with support for several Arduino Shields. Those VIs can be found as part of the product function palette (refer to the Product User Manual for more information on how to access the product function palette). Starting in version 1.0.0.17 of the Compiler, users can port existing Arduino Libraries and create their own LabVIEW API VIs to bring a library's functionality into LabVIEW. This allows the community to take advantage of the existing ecosystem of Arduino shields and to create even more complex applications in LabVIEW for the Arduino platform.

Support for this feature is community based at [www.geverywhere.com](http://www.geverywhere.com) as we encourage sharing of the created API VIs with the community. This document will present the process for one to port an existing Arduino Library through an example. We will port the library for the Analog Shield by Diligent (<http://www.digilentinc.com/Products/Detail.cfm?NavPath=2,648,1261&Prod=TI-ANALOG-SHIELD>).

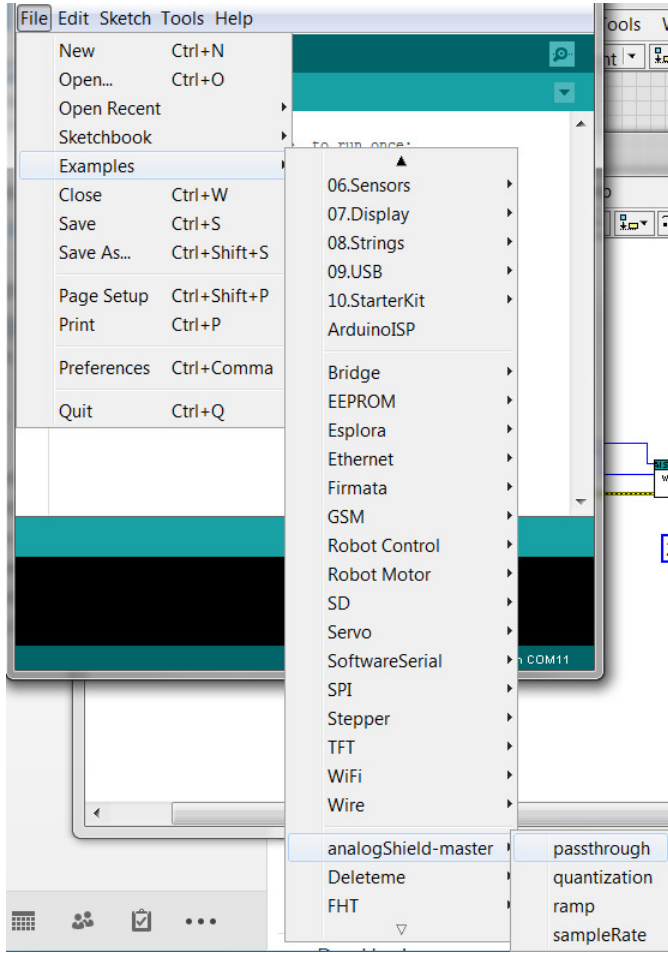
It is assumed at least a medium level of familiarity with the C language by the developer as example Arduino code that is part of the library will need to be read and understood as well as some lines of C-code will have to be written by the

developer as part of the process.

#### Step 1 – Installing the Arduino Library

The first step in the process is the installation of the library the developer wishes to port into the LabVIEW Compiler. In our case, the latest library can be found at <https://github.com/mwingerson/analogShield>. You will need to open the Arduino IDE and install the desired library. The following article <http://www.arduino.cc/en/Guide/Libraries> provides detailed information on how to perform a library installation on your Arduino IDE.

Once the library has been installed, you should be able to open one of the library's shipping examples from the Arduino IDE. To do that, go to File->Examples, as illustrated below.



If you are unable to see the examples for the library you just installed, close and re-open the Arduino IDE. If that still doesn't solve the issue, the installation of the library was not successful and the process above will need to be repeated.

The last step to verify the library is functional with your hardware is to actually run one of its shipping examples. Since the libraries are usually open source, it is important to make sure they work as expected with the hardware, in the Arduino IDE, prior to porting it to the Arduino Compatible Compiler for LabVIEW. It is important not to skip any of the described tasks, as they may save you a lot of time in the end.

#### Step 2 – Identifying the Functions to Port

Once the step described above is completed, the next step is to identify which of the library's functions will be ported to the Arduino Compatible Compiler for LabVIEW. This step requires familiarity with C-code as the library's shipping examples code will need to be understood.

The main idea behind this step is to identify which of the library's functions are API functions and which ones are internal functions used by the API functions. You don't necessarily need to port all library's functions to LabVIEW, though you could, but porting just the main API functions will make for a cleaner and easier to understand and utilize the LabVIEW library.

Since we are using Digilent's Analog Shield on this User Guide, let's investigate the first shipping example from the list - passthrough. Here it is the source code of the example:

```
#include <analogShield.h> //Include to use analog shield.
#include <SPI.h> //required for ChipKIT but does not affect Arduino

void setup()
{
    //no setup
}

unsigned int count = 0;
```

```
void loop()
{
    count = analog.read(0); //read in on port labeled 'INO'
    analog.write(0, count); //write out the value on port labeled 'OUT0'
}
```

The code above basically reads analog input channel 0, stores the value into a variable named "count" and write the value stored in "count" to analog output channel 0. This example shows very two API functions very clearly; analog.read() and analog.write().

Let's take a look at the second shipping example from the list; quantization. The source code of the example follows:

```
#include <analogShield.h> //Include to use analog shield.
#include <SPI.h> //required for ChipKIT but does not affect Arduino

void setup()
{
    //no setup
}

unsigned int value, Out0, Out1, Out2, Out3;
void loop()
{
    //Read the signal in from port IN0, store in 'value'
    value = analog.read(0);
    // Output quantization:
    // "AND" the 16-bit data with masks containing ONES for bits of the original that will pass through
    // while ZEROES for bits remove information for those bits, effectively reducing the ADC resolution
    Out0 = value; // Out 0 = 16 bit, full resolution (2^16 = 65,536 quantization levels)
    Out1 = value&0xFC00; // Out 1 = 6 bit (2^6 = 64 quantization levels)
    Out2 = value&0xF000; // Out 2 = 4 bit (2^4 = 16 quantization levels)
    Out3 = value&0xC000; // Out 3 = 2 bit (2^2 = 4 quantization levels)
    //write the data back out! Remember, writing 4 channels out takes nearly 4x as long!
    analog.write(Out0,Out1,Out2,Out3, true); //takes ~50uS
    //analog.write(Out1); //faster one channel -- reading is the longer part, but this only takes 30uS
}
```

The code above executes an analog read and operates on the value read, creating four other variables Out0 through Out3. Each of these variables is passed as a parameter to the analog.write() function. One can see on this example that the same analog.read() was used, but that the analog.write() was used with a different construct. In this example, the analog.write() function receives four values and a boolean parameter as inputs, whereas in the previous example, it received a channel number and a value as parameters. API functions with different constructs can have each of its multiple constructs made into a separate LabVIEW API VI, as we will see a little later on this Guide.

Now that we have evaluated two examples, it is probably a good time to take a look at the library source code itself. There are usually two files of interest in a library, a ".h" and a ".c" or ".cpp" files. Explanation of what these files are is beyond the scope of this guide as it is related to the C language. Open the ".c" file. In our case, it is file named "analogShield.cpp".

The ".c" file will have the implementation of the API functions that we will be porting to LabVIEW. The implementation is important as we will need to extract the list of parameters and their corresponding data types so we can map our API VIs connector pane to match the same parameters list.

In looking at the analogShield.cpp file, it is clear that the following functions are API functions and will be ported to LabVIEW: Read, Signed Read and Write. Now, there is a caveat with the Write function. Remember that we saw two different constructs for the Write function on the two examples we have analyzed. In fact, in looking through the library source code, the Write function has four different constructs: A single channel write, two channel write, three channel write and four channel write. As it was mentioned above, each one of these constructs will be made into a separate API VI. Therefore, each construct's parameter list will need to be understood.

Also, since LabVIEW usually has an Open and Close VIs where initialization and clean up code usually reside respectively, it is good practice to include an Open and Close API VIs for consistency. It will become clear what code would be placed in Open and Close a little later in this guide once we turn our attention to the LabVIEW side of the port.

In summary, the list of API VIs for the Diligent Analog Shield will be: Open, Read, Signed Read, Write, Write Two Channels, Write Three Channels, Write Four Channels and Close.

One last task prior to calling this step complete is to determine the parameters list for each of the API functions as these parameters will need to be mapped by the LabVIEW API VIs themselves. The list of parameters can be extracted from the ".c" or ".h" file, depending on where the API function implementation is coded.

In the case of our example, the functions implementations are in "analogShield.cpp". The list below shows the functions prototypes extracted from the ".cpp" file that contains the parameter list for each one of the functions that we elected to port to LabVIEW.

```
unsigned int analogShield::read(int channel, bool mode)
int analogShield::signedRead(int channel, bool mode)
void analogShield::write(int channel, unsigned int value) //This is the construct used for Write API VI
void analogShield::write(unsigned int value0, unsigned int value1, bool simul)//This is the construct used for Write Two Channels API VI
void analogShield::write(unsigned int value0, unsigned int value1, unsigned int value2, bool simul)// This is the construct used for Write Three Channels API VI
```

```
void analogShield::write(unsigned int value0, unsigned int value1, unsigned int value2, unsigned int value3, bool simul)//
This is the construct used for Write Four Channels API VI
```

At this point, we have all we need in order to start our work in LabVIEW.

#### Step 3 – Creating the LabVIEW API VIs

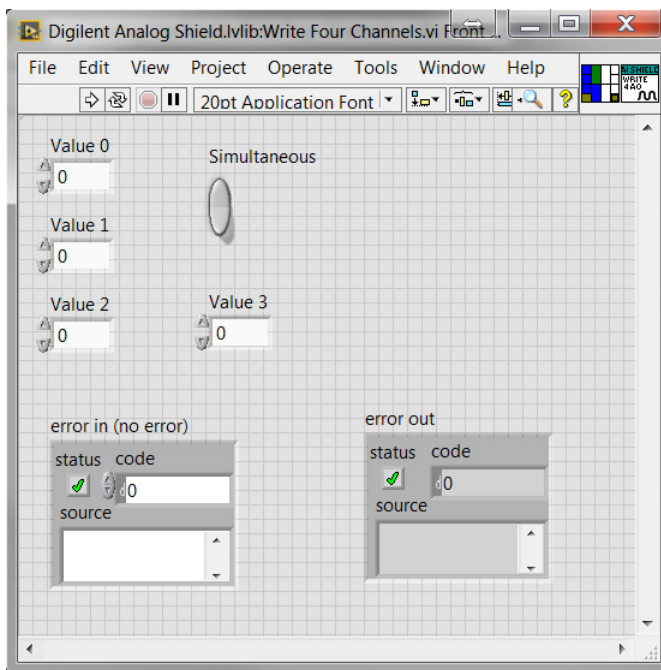
Now that we have determined the list of all API VIs that will be created as well as their respective parameter list, it is time to actually create them. The way the Arduino Compatible Compiler identifies that a VI is an Arduino primitive (or, in our case, an Arduino Library) VI, is through password protection. If a VI is password protected, the Arduino Compatible Compiler for LabVIEW will not attempt to interpret the content of the subVI itself, and will expect to see an implementation for the subVI in C-code.

Since the work we are doing is porting existing C-code from an Arduino library so it gets bundled along with the other code that is generated from LabVIEW primitives, each of the LabVIEW API VIs will need to be password protected.

We need to create a LabVIEW API VI per function API that we have defined in step 2. Moreover, each of the LabVIEW API VIs will have to have a connector pane that matches the parameter list of the API function as well as the parameters datatypes. Let's take an example to illustrate this statement. Take the Write Four Channels API VI; its function prototype is repeated here for convenience:

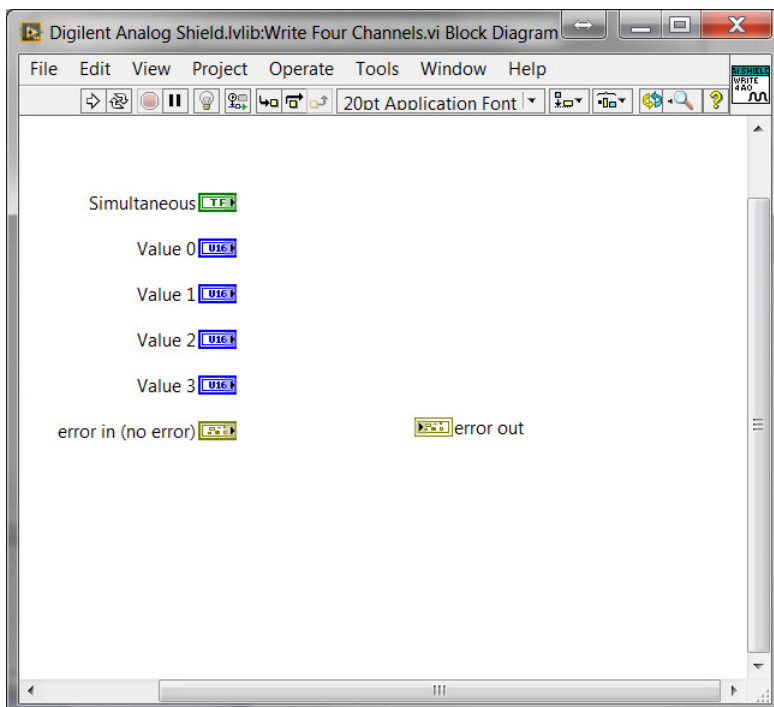
```
void analogShield::write(unsigned int value0, unsigned int value1, unsigned int value2, unsigned int value3, bool simul)//
This is the construct used for Write Four Channels API VI
```

The function prototype states that the function needs four unsigned int parameters and one boolean parameter, and that the function doesn't return anything, i.e., no outputs only inputs. We need to create a password protected LabVIEW VI that has the same matching elements of the original function prototype. The password used for the Diligent APIs is **template**. The figure below shows the concept.



A quick look at the VI block diagram shows that the API VI is really only a shell mapping the inputs and output parameters for the API function it ports. It is important to see how the data type for each of the controls match the ones declared in the C function prototype. Another point to highlight is that the error clusters are part of the connector pane as well, even though the original C function doesn't take an error argument.

As it is stressed in the Arduino Compatible Compiler user manual, error clusters are only used for data flow purposes and are ignored by the compiler. Therefore, it is important to make sure your LabVIEW API VIs include the error input/output connectors so your VIs can offer the same feature of allowing data flow to be controlled via the error cluster.



We need to create other LabVIEW API VIs, to map the other functions from our list. We will omit that process in this guide as they are a repetition of this first case. You can find the other API VIs for the Digilent Analog Shield at "[LABVIEWDIR]\vi.lib\Aledyne-TSXperts\Arduino Compatible Compiler for LabVIEW\addons\Digilent Analog Shield" within the Arduino Compatible Compiler for LabVIEW installation. It is a good exercise to open each LabVIEW API VI and make sure you understand the mapping of their parameters from the connector pane back to the original C function.

**IMPORTANT:** It is extremely important to point out that, like the Digilent Analog Shield set of API VIs that were created to illustrate how existing Arduino libraries can be ported into the Arduino Compatible Compiler for LabVIEW, any user created set of API VIs need to reside in the directory: "[LABVIEWDIR]\vi.lib\Aledyne-TSXperts\Arduino Compatible Compiler for LabVIEW\addons". This indicates to the compiler that those VIs are user created, and they corresponding C-code can be displayed by the Arduino Code Display Tool, explained in step 5 below.

#### Step 4 – Using the LabVIEW Library Template

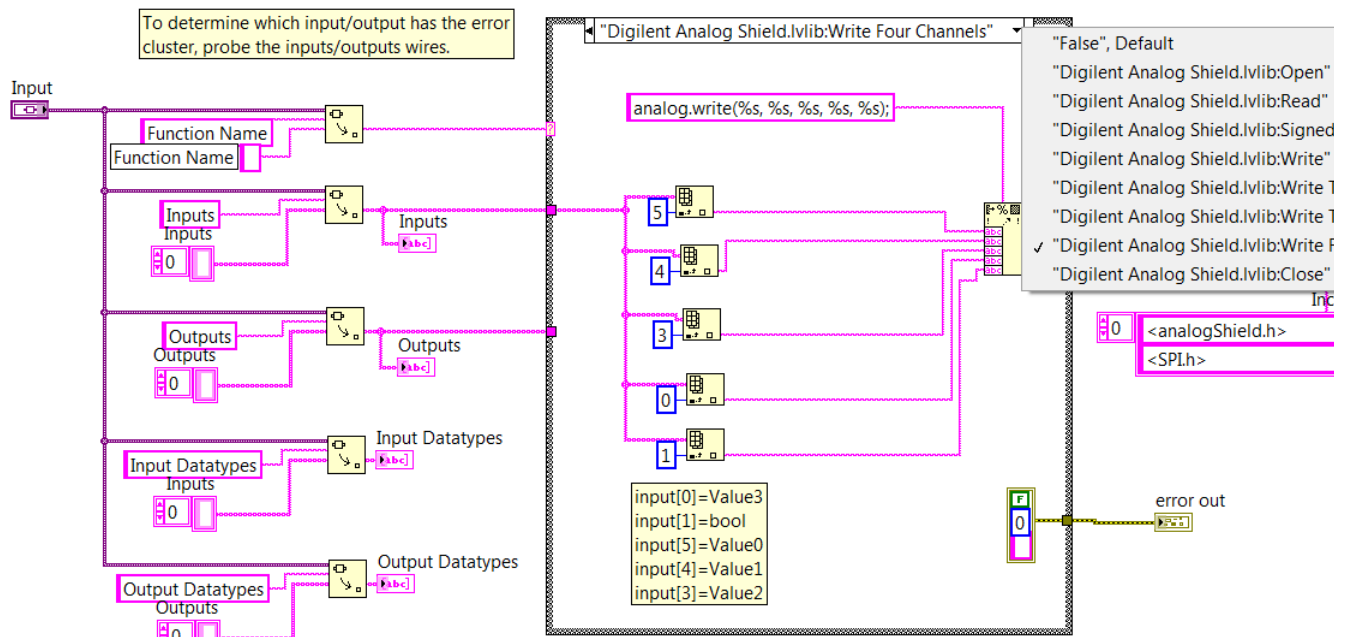
The Arduino Compatible Compiler for LabVIEW product ships with a library template, located at "[LABVIEWDIR]\vi.lib\Aledyne-TSXperts\Arduino Compatible Compiler for LabVIEW\addons\Template". This template was created to simplify the task of porting Arduino Libraries to work with the Arduino Compatible Compiler for LabVIEW.

The template contains the following components:

- LabVIEW Library: This is the lvlib that will contain all your API VIs.
- Translator.vi: This is where all the magic happens. This is where you will map each of the API VIs parameter list, add the needed list of includes and defines as well as add any C-code needed for a given API VI.
- Add and Subtract: These are two API VIs created as an example. They are password protected such that the Compiler recognizes them as Arduino APIs. The password is **template**.
- Test.vi: This VI is used to test the implementation of the Add and Subtract API VIs.
- libraries: This is where the Arduino library can be copied or any custom ".c" and ".h" files with the specific C implementation. The Arduino Compatible Compiler for LabVIEW will automatically install any libraries stored here such that Step 1 of this guide is not necessarily required. Although it is still good practice to execute Step 1 so that you can ensure the library is working with hardware within the Arduino IDE first. For the case of the template, there is one ".h" file with the implementation of the functions to be ported by the Add and Subtract API VIs.

As it was mentioned above, the Translator.vi is where the mapping actually happens. Let's take an example to illustrate how the mapping is actually done. We have created in the previous step the LabVIEW API VI Write Four Channels.vi; which maps the original C function construct that takes five input parameters.

One can start with the Translator.vi from the Template provided. Then, on the central case statement, one would need to create a separate case for each LabVIEW API VI from your library. In this example, the Template has been saved as a separate lvlib named Digilent Analog Shield and inside its Translator.vi, a case for each of the LabVIEW API VIs defined in the list has been created, as illustrated below.



The mapping of the input and output parameters will happen inside each case frame, in the way to match the correct LabVIEW controls and indicators wired to the VI connector pane, to the intended corresponding C function parameter in the function prototype.

Take the figure above for instance. The Write Four Channels case frame is shown. As you can see in the LabVIEW code in the case frame, the Compiler is told that once it sees this LabVIEW VI, it will use the prototype that is coded inside the case frame. The following call needs to be replicated:

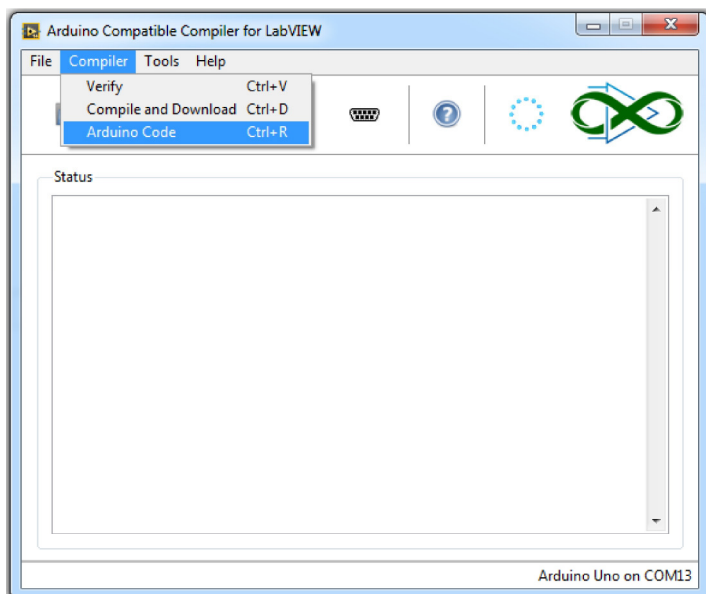
```
analog.write(unsigned int value0, unsigned int value1, unsigned int value2, unsigned int value3, bool simul);
```

We need to have "analog.write(%s, %s, %s, %s, %s, %s);" inside the case frame, and each of the "%s" instance will be replaced by the Format to String primitive LabVIEW VI with an element from the incoming "Inputs" array. Since the analog.write doesn't return an output, and an output in the LabVIEW API VI connector pane is not mapped, we don't need to worry about the incoming "Outputs" array.

A common question may be "How can I know which array element will map to each parameter from my C function parameter list?". To answer that question, we would like to introduce a new tool that was added to the Arduino Compatible Compiler for LabVIEW in version 1.0.0.17; the Arduino Code Display Tool, which is a topic of the next step.

#### Step 5 – The Arduino Code Display Tool

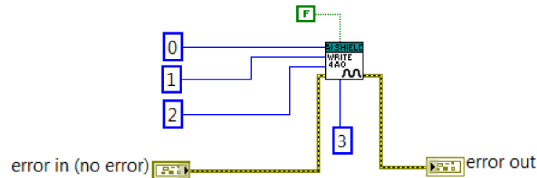
In order to make your task of porting an Arduino library easier, we have introduced the Arduino Code Display tool in the Arduino Compatible Compiler for LabVIEW version 1.0.0.17 and beyond. You can access the tool from the Compiler menu, as illustrated below.



The process of viewing the Arduino Code that is generated by your library port is similar to the one of compiling an



Arduino VI. You will need to first create a test VI, add the LabVIEW API VI you are interested in seeing the generated Arduino code for, populate all the VI inputs and outputs and save the test VI. The figure below shows a test VI created for the Write Four Channels VI we have been working through in this guide.



Then open the Arduino Compatible Compiler for LabVIEW and load the test VI you created, as if you were going to compile it. Then navigate to the Arduino Code menu item. Once you select that menu option, another window will open displaying the C-code generated for the test VI.

```

#include "LVArray.h"
#include "A4Lhelper.h"
#include <analogShield.h>
#include <SPI.h>
void setup()
{
    unsigned short Const296;
    unsigned short Const244;
    unsigned short Const198;
    unsigned short Const152;
    boolean Const127;
    Const296 = 3;
    Const244 = 2;
    Const198 = 1;
    Const152 = 0;
    Const127 = false;
    analog.write(Const152, Const198, Const244, Const296, Const127);
}

void loop()
{
}

```

As you can see in the C-code snippet above, the tool outputs for you all declared variables as well as the current mapping of parameters based on the elements you have selected from the "Inputs" array in the case frame for the Write Four Channels API VI inside the Translator.vi.

Notice how we purposely numbered the unsigned integer constants wired to the inputs so we can easily tell which one shows up in what element in the function call. The order showing above is correct as Value 0 in LabVIEW is Const152, which correct shows up as the first parameter in the list. Value 1 is Const198, Value 2 is Const244, Value 3 is Const296 and simultaneous is Const127. Should one or more of these parameters show in the wrong order in the parameters list, you would be able to spot the mistake, go back to the Translator.vi and change the index of the Inputs array to match the correct parameters order.

One quick test you can do once you are happy with the mapping of the LabVIEW API VI parameters is to use the Compile button in the Arduino Compatible Compiler for LabVIEW for the test VI you just displayed the Arduino Code for. This will tell you if whatever C code you created inside the case frame for that specific API VI inside the Translator.vi is properly compiling. If the Arduino Compatible Compiler for LabVIEW generates a compilation error, you know you need to review whatever C code you have inside the case frame.

Furthermore, you can also copy this code from the LabVIEW screen presented and paste it directly in the Arduino IDE if you would like to experiment with the code in C within the Arduino IDE.

A special case to be aware of is the use of array datatypes as inputs or outputs. This gets more advanced since the LabVIEW array implementation is handled in a C++ class, located in "[LABVIEWDIR]\vi.lib\Aledyne-TSXperts\Arduino Compatible Compiler for LabVIEW\Arduino Libraries\LVArray\LVArray.h". Common needs may be to index a certain element of an array, get the size of the array, or obtain a pointer to the array within the C code in the Translator.vi. For example if an input array is called **Array123**, one can index the 10th element of that array using the syntax `Array123[10]`. One can obtain the size of the array with the syntax `Array123.size()` and can obtain a C pointer to the array using `Array123.ptr()`. These may be required in order to pass array data to C functions that expect it. You can refer to **LVArray.h** for a complete list of available array C++ methods.

One very important thing to point out is that the Arduino Code Tool will only work for user created library API VIs and will generate an error if any LabVIEW primitive is included as part of the test VI. Therefore, make sure the test VI you create is used only to map the API VI parameters, as explained in this guide.

#### Step 6 – Testing the API VIs

Once you have completed the process above for all LabVIEW API VIs and all of them are compiling without errors, the best way to make sure your implementation is sound is to replicate the shipping examples from the Arduino Library, but using your newly created LabVIEW API VIs and the Arduino Compatible Compiler for LabVIEW.

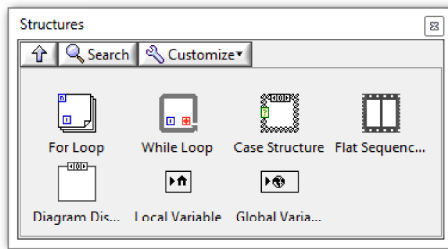
Make sure you replicate the exact functionality the library C code example does, and compare the results. Once they match for the example and you exercise all functions of your library, you are done and your library is ready for

distribution.

©2015 TSXperts/Aledyne. All rights reserved

## Structures Palette

Installed With: Arduino Compatible Compiler for LabVIEW



The Structures palette contains the following LabVIEW primitives. Their functionality when deployed onto an Arduino target is identical to that of LabVIEW unless otherwise specified. Using any structure outside this palette will result in a compilation error.

### -For Loop

NOTE: It is not advised to use auto-indexing outputs as this can result in memory starvation and forces reallocation of memory, which can take excessive processor cycles for small Arduino targets. It is also recommended to minimize the use of shift registers, and do not use them for large arrays, as they use excessive memory as multiple copies are required to implement shift registers.

### -While Loop

NOTE: It is not advised to use auto-indexing outputs as this can result in memory starvation and forces reallocation of memory, which can take excessive processor cycles for small Arduino targets. It is also recommended to minimize the use of shift registers, and do not use them for large arrays, as they use excessive memory as multiple copies are required to implement shift registers.

### -Case Structure

NOTE: outputs of case structures cannot be set to "Use Default If Unwired". All outputs must be wired, otherwise a compilation error will result.

### -Flat Sequence

### -Diagram Disable

NOTE: Any code inside of a diagram disable structure will not be compiled and deployed to the Arduino target.

### -Local Variable

NOTE: Local Variables will be implemented as global variables on the Arduino target so its usage should be limited.

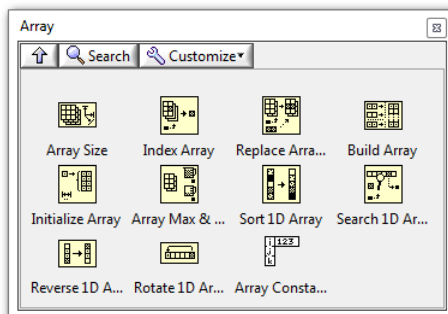
### -Global Variables

NOTE: These are the only way to share data between an interrupt and main VI when using interrupts on an Arduino target. Beware that there must be a writer of the global somewhere in the project as writing to a global is what triggers creation of the data variable. If a global is created and only read within the code, a compilation error will occur.

©2015 TSXperts/Aledyne. All rights reserved

## Array Palette

Installed With: Arduino Compatible Compiler for LabVIEW



The Array palette contains the following LabVIEW primitives. Their functionality when deployed onto an Arduino target is identical to that of LabVIEW unless otherwise specified. Using any array primitive outside this palette will result in a compilation error. **Only 1-D Arrays** are supported at this time. It is **strongly** recommended NOT to use array functions on small Arduino targets, like the Arduino Uno, which only has 2k of RAM. Arrays are implemented using dynamic memory allocation, so it is not known until there is a runtime error if memory has been exhausted. In this case, the program may start to act irregular, and there is no deterministic way to verify if memory has been exceeded. The Check Unused RAM utility VI may be used to estimate available RAM while a program is running to determine free space.

### -Array Size

### -Index Array

### -Replace Array Subset

## -Build Array

NOTE: It is **strongly** recommended NOT to use this function as it will be slow and can easily starve the memory of Arduino targets due to low memory availability and dynamic memory allocation. Also, since 2-D arrays are not supported, concatenate inputs can only be used with this function.

## -Initialize Array

## -Array Max & Min

## -Sort 1D Array

## -Search 1D Array

## -Reverse 1D Array

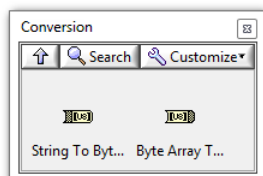
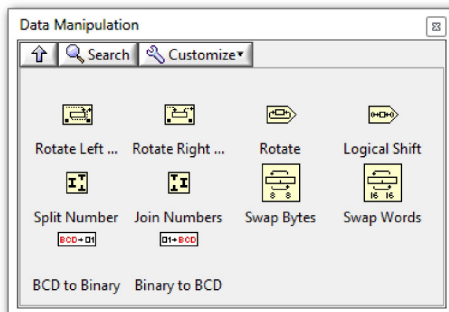
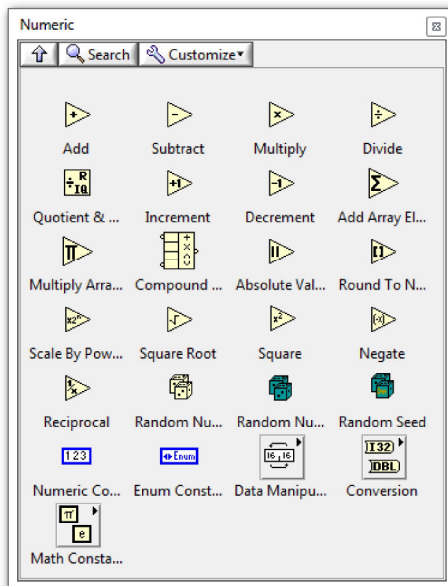
## -Rotate 1D Array

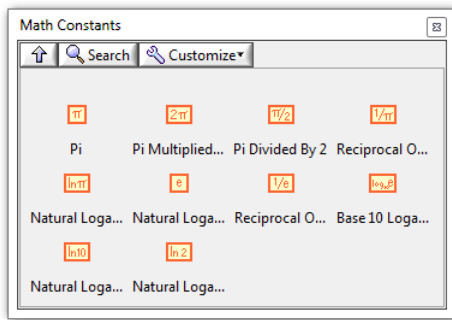
## -Array Constant

©2015 TSXperts/Aledyne. All rights reserved

## Numeric Palette

Installed With: Arduino Compatible Compiler for LabVIEW





The Numeric palette contains the following LabVIEW primitives. Their functionality when deployed onto an Arduino target is identical to that of LabVIEW unless otherwise specified. Using any numeric primitive outside this palette will result in a compilation error.

- Add
- Subtract
- Multiply
- Divide
- Quotient & Remainder
- Increment/Decrement

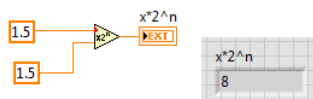
NOTE: using this on enumerated types does not wrap around like in native LabVIEW. For example, if the last item in an enumerated list has a value of 3 and it is incremented, native LabVIEW will output a value of 0 but the Arduino Compatible Compiler for LabVIEW will output a value of 4. The same is true for decrementing if the enumerated value is 0. It is recommended not to use enumerated types with numeric functions.

- Add/Multiply Array Elements
- Compound Arithmetic

NOTE: Invert inputs and outputs are not supported. Use external boolean "Not" primitives instead.

- Absolute Value
- Round To Nearest
- Scale By Power Of Two

NOTE: This function has a bug when running in LabVIEW. For instance, writing a floating point number half way between two integers to both inputs will cause an incorrect result in LabVIEW. In this example, 1.5 wired to the 'n' input should be rounded up to 2 and the result should be  $1.5 * 2^2$  or 6. But LabVIEW will round both inputs giving  $2 * 2^2$  or 8. This function has been implemented in the correct way on the Arduino Compatible Compiler for LabVIEW, so this example would produce a value of 6 on an Arduino target. Wiring an integer to the 'n' input is more of a common application though.



- Square Root
- Square
- Negate
- Reciprocal
- Random Number (0-1)
- Numeric/Enum Constant
- Rotate Left/Right
- Rotate
- Logical Shift
- Split Numbers
- Join Numbers
- Swap Bytes/Words
- String to Byte Array
- Byte Array to String

NOTE: Any NULL (0x00) bytes in an array cannot be represented in Arduino language and will cause the string to be prematurely terminated during conversion. Do not use this function to convert NULL characters.

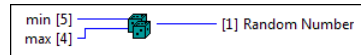
- Math Constants

©2015 TSXperts/Aledyne. All rights reserved

## Random Number In Range VI

**Installed With:** Arduino Compatible Compiler for LabVIEW

Generates pseudo-random numbers in the specified range between **min** and **max**.



**min** is the lower bound of the random value, inclusive.

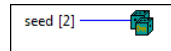
**max** is the upper bound of the random value, exclusive.

**Random Number** is a random number between min and max-1.

## Random Seed VI

**Installed With:** Arduino Compatible Compiler for LabVIEW

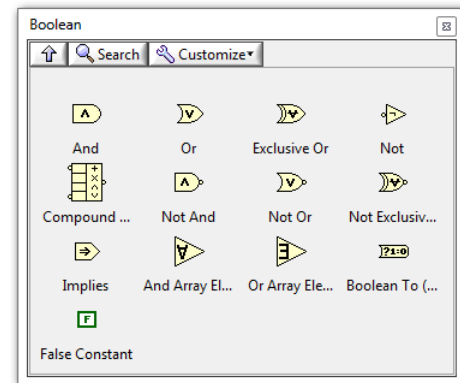
Initializes the Arduino pseudo-random number generator, causing it to start at an arbitrary point in its random sequence by setting **seed**. This sequence, while very long, and random, is always the same. If it is important for a sequence of values generated by Random Number In Range.vi to differ, on subsequent deployments of a program, use Random Seed.vi to initialize the random number generator with a fairly random input, such as Analog Read.vi on an unconnected pin. Conversely, it can occasionally be useful to use pseudo-random sequences that repeat exactly. This can be accomplished by calling Random Seed.vi with a fixed number, before starting the random sequence.



**seed** to initialize the pseudo-random number generator.

## Boolean Palette

**Installed With:** Arduino Compatible Compiler for LabVIEW



The Boolean palette contains the following LabVIEW primitives. Their functionality when deployed onto an Arduino target is identical to that of LabVIEW unless otherwise specified. Using any boolean primitive outside this palette will result in a compilation error.

-And/Or/Exclusive Or/Not

-Compound Arithmetic

NOTE: Invert inputs and outputs are not supported. Use external boolean "Not" primitives instead.

-Not And/Not Or/Not Exclusive Or

-Implies

-And/Or Array Elements

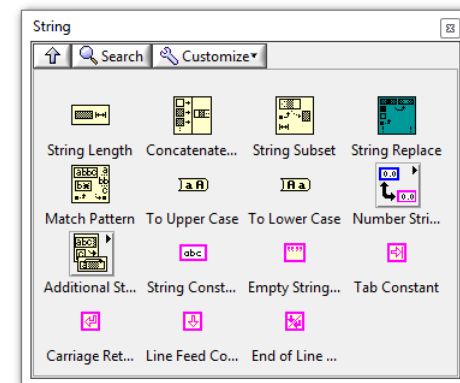
-Boolean to (0,1)

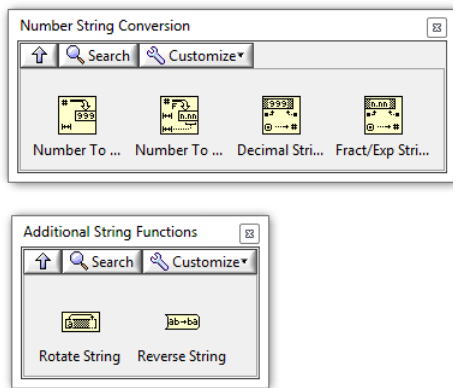
-True/False Constant

©2015 TSXperts/Aledyne. All rights reserved

## String Palette

**Installed With:** Arduino Compatible Compiler for LabVIEW





The String palette contains the following LabVIEW primitives. Their functionality when deployed onto an Arduino target is identical to that of LabVIEW unless otherwise specified. Using any string primitive outside this palette will result in a compilation error. It is **strongly** recommended NOT to use string functions on small Arduino targets, like the Arduino Uno, which only has 2k of RAM. Strings are implemented using dynamic memory allocation, so it is not known until there is a runtime error if memory has been exhausted. In this case, the program may start to act irregular, and there is no deterministic way to verify if memory has been exceeded. The Check Unused RAM utility VI may be used to estimate available RAM while a program is running to determine free space.

- String Length
- Concatenate String
- String Subset
- Match Pattern
- To Upper Case
- To Lower Case
- Number to Decimal String
- Number to Fractional String
- Decimal String to Number

NOTE: offset past number output always returns 0. The default input can only be used to assign the output datatype. The default value will not be used in the event a conversion cannot take place. Also, if the converted value exceeds the size of the output datatype, the value will not be clamped like in LabVIEW desktop, it will rollover (e.g. if the input String is "256" and the default type is set to U8, the output will be 0, not 255).

- Fract/Exp String to Number

NOTE: System decimal point inputs are ignored. Offset past number output always returns 0. The default input can only be used to assign the output datatype. The default value will not be used in the event a conversion cannot take place. Also, if the converted value exceeds the size of the output datatype, the value will not be clamped like in LabVIEW desktop, it will rollover (e.g. if the input String is "256.0" and the default type is set to U8, the output will be 0, not 255).

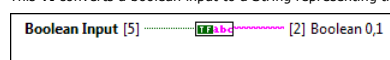
- Rotate String
- Reverse String

©2015 TSXperts/Aledyne. All rights reserved

## Boolean to String VI

**Installed With:** Arduino Compatible Compiler for LabVIEW

This VI converts a boolean input to a String representing the boolean value as either a 0 or 1.



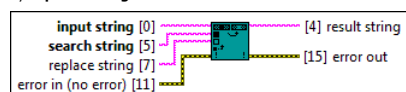
**Boolean Input** is a boolean to convert to String.

**Boolean 0,1** is a the converted String representation of the boolean input and is either "0" or "1".

## String Replace VI

**Installed With:** Arduino Compatible Compiler for LabVIEW

This VI allows you to replace all instances of a given character with another character. You can also use replace to replace substrings of a string with a different substring. The implementation is similar to the native LabVIEW search and replace string, but with limited capabilities on the Arduino target. A single search string can be entered in **search string** and all occurrences of it in **input string** will be replaced by **replace string**.



**input string** specifies the input string you want the function to search.

**search string** specifies the string you want to search for and replace.

**replace string** specifies the string you want to insert in place of search string.

**error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error. NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.



**TF** **status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**I32** **code** is the error or warning code. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**abc** **source** describes the origin of the error or warning. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**abc** **result string** contains the input string with all occurrences of search string replaced with replace string.

**TF** **error out** passes error or warning information out of a VI to be used by other VIs. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

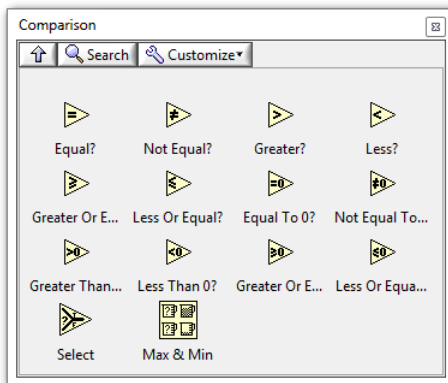
**TF** **status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**I32** **code** is the error or warning code. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**abc** **source** string describes the origin of the error or warning. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

## Comparison Palette

Installed With: Arduino Compatible Compiler for LabVIEW



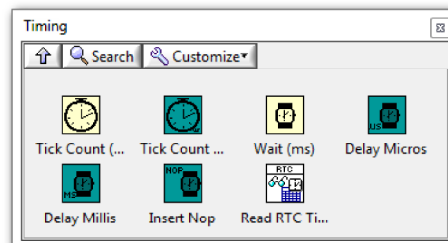
The Comparison palette contains the following LabVIEW primitives. Their functionality when deployed onto an Arduino target is identical to that of LabVIEW unless otherwise specified. Using any Comparison primitive outside this palette will result in a compilation error.

- Equal?/Not Equal?
- Greater?/Less?
- Greater Or Equal/Less Or Equal?
- Equal to 0?/Not Equal To 0?
- Greater Than 0/Less Than 0?
- Greater Or Equal to 0/Less Or Equal To 0?
- Select
- Max & Min

©2015 TSXperts/Aledyne. All rights reserved

## Timing Palette

Installed With: Arduino Compatible Compiler for LabVIEW



The Timing palette contains the following LabVIEW primitives:

- Tick Count (ms) - NOTE: tick count resets to 0 when target is reset.

-Wait (ms)

Their functionality when deployed onto an Arduino target is identical to that of LabVIEW unless otherwise specified. In addition, higher resolution timers and wait functions exist for micro second resolution.

NOTE: for tick count functions, the time reported is ticks from when the target started running. Since the targets do not contain an on board real time clock, the timer count will reset when the program is restarted. Using any timing primitive outside this palette will result in a compilation error.

©2015 TSXperts/Aledyne. All rights reserved

## Delay Millis VI

**Installed With:** Arduino Compatible Compiler for LabVIEW

Pauses the program for the amount of time (in milliseconds) specified in **delay**. This is similar to the Wait (ms) LabVIEW primitive, however, error in and out terminals are included here for flow control.



**delay** in microseconds to halt program execution. The largest value that will produce an accurate delay is 16383.

**error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**code** is the error or warning code. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**source** describes the origin of the error or warning. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**error out** passes error or warning information out of a VI to be used by other VIs. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**code** is the error or warning code. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

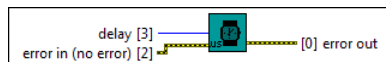
**source** string describes the origin of the error or warning. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

## Delay Micros VI

**Installed With:** Arduino Compatible Compiler for LabVIEW

Pauses the program for the amount of time (in microseconds) specified in **delay**. There are a thousand microseconds in a millisecond, and a million microseconds in a second. Currently, the largest value that will produce an accurate delay is 16383. This could change in future Arduino releases. For delays longer than a few thousand microseconds, you should use the native LabVIEW Wait (ms) instead.



**delay** in microseconds to halt program execution. The largest value that will produce an accurate delay is 16383.

**error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**code** is the error or warning code. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**source** describes the origin of the error or warning. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**error out** passes error or warning information out of a VI to be used by other VIs. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**code** is the error or warning code. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

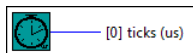
**source** string describes the origin of the error or warning. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

## Tick Count Micros VI

**Installed With:** Arduino Compatible Compiler for LabVIEW

Returns the number of microseconds in **ticks (us)** since the Arduino board began running the current program. This number will overflow (go back to zero), after approximately 70 minutes. On 16 MHz Arduino boards (e.g. Duemilanove and Nano), this function has a resolution of four microseconds (i.e. the value returned is always a multiple of four). On 8 MHz Arduino boards (e.g. the LilyPad), this function has a resolution of eight microseconds. Note: there are 1,000 microseconds in a millisecond and 1,000,000 microseconds in a second.

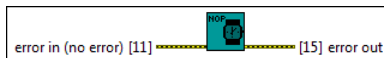


**ticks (us)** is the number of microseconds since the Arduino board began running the current program.

## Insert Nop VI

**Installed With:** Arduino Compatible Compiler for LabVIEW

Inserts a nop (no operation) instruction to induce a very short delay of one instruction. The delay is based on the CPU architecture. For example, on the ATmega168 running at 16MHz, each nop statement executes in one machine cycle yielding a 62.5 ns (nanosecond) delay.



**error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**code** is the error or warning code. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**source** describes the origin of the error or warning. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**error out** passes error or warning information out of a VI to be used by other VIs. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**code** is the error or warning code. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

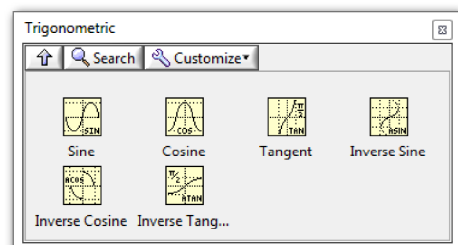
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**source** string describes the origin of the error or warning. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

## Trigonometric Palette

**Installed With:** Arduino Compatible Compiler for LabVIEW



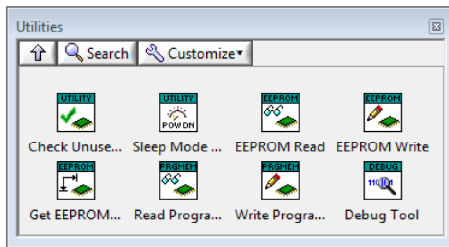
The Trigonometric palette contains the following LabVIEW primitives. Their functionality when deployed onto an Arduino target is identical to that of LabVIEW unless otherwise specified. Using any trigonometric primitive outside this palette will result in a compilation error.

- Sine
- Cosine
- Tangent
- Inverse Sine
- Inverse Cosine
- Inverse Tangent

©2015 TSXperts/Aledyne. All rights reserved

## Utilities Palette

**Installed With:** Arduino Compatible Compiler for LabVIEW



The utilities palette contains utility APIs for interfacing with advanced modes of the Arduino target.

©2015 TSXperts/Aledyne. All rights reserved

## Check Unused RAM VI

**Installed With:** Arduino Compatible Compiler for LabVIEW

Reports the space between the heap and the stack, which is the amount of dynamic memory available in the system. It does not report any de-allocated memory that is buried in the heap. Buried heap space is not usable by the stack, and may be fragmented enough that it is not usable for many heap allocations either. The space between the heap and the stack is what you really need to monitor if you are trying to avoid stack crashes.



**error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**code** is the error or warning code. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**source** describes the origin of the error or warning. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**Free RAM** is the amount of memory space in bytes available between the heap and the stack.

**error out** passes error or warning information out of a VI to be used by other VIs. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**code** is the error or warning code. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**source** string describes the origin of the error or warning. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

## Sleep Mode Power Down VI

**Installed With:** Arduino Compatible Compiler for LabVIEW

Puts the microcontroller to sleep using the SLEEP\_MODE\_PWR\_DOWN option for avr. This uses the least power but turns almost everything off, so your options for wake interrupts and the like are limited. Waking up the microcontroller can be done via external interrupt (change in pin state), however only a LOW state can take it out of sleep. The currently executing code will stop at this VI until it wakes from sleep, at which point execution will resume after this VI.



**error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**code** is the error or warning code. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**source** describes the origin of the error or warning. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**error out** passes error or warning information out of a VI to be used by other VIs. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**code** is the error or warning code. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about



## Read Program Memory VI

**Installed With:** Arduino Compatible Compiler for LabVIEW

Reads data previously stored in flash (program) memory. The data previously stored in program memory is only available as read-only to your program through this function. This is a good way to get use of more memory for read-only data that is known at compile time like lookup tables. NOTE: The input array **Name** must be a control or constant and must be directly wired to this function.



**Name** defines the unique identifier to the data saved in Program Memory.

**Index** defines the index of the data to read from in Program Memory.

**error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**code** is the error or warning code. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**source** describes the origin of the error or warning. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**Data** returns the data read from the specified index, **Idx**, of the data array previously stored in Program Memory with the specified **Name**.

**error out** passes error or warning information out of a VI to be used by other VIs. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**code** is the error or warning code. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

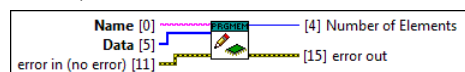
**source** string describes the origin of the error or warning. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

## Write Program Memory VI

**Installed With:** Arduino Compatible Compiler for LabVIEW

Store data in flash (program) memory instead of SRAM. This function tells the compiler to put the input array into flash memory instead of into SRAM, where it would normally go. Therefore, the data stored in program memory is only available as read-only to your program. This is a good way to get use of more memory for read-only data that is known at compile time like lookup tables. You must use the Read Program Memory functions to retrieve the data from your program at run-time. NOTE: The input array must be a control or constant and must be directly wired to this function as well as the Name to assign to the array.



**Name** defines the unique identifier to the data saved in Program Memory.

**Data** is the array of data to write to Program Memory.

**error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**code** is the error or warning code. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**source** describes the origin of the error or warning. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**Number of Elements** stored in Program Memory.

**error out** passes error or warning information out of a VI to be used by other VIs. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**code** is the error or warning code. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**source** string describes the origin of the error or warning. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

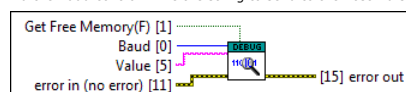
## Debug Tool VI

**Installed With:** Arduino Compatible Compiler for LabVIEW

The Debug Tool is a VI that can be placed into existing Arduino LabVIEW code to aid in debugging. The Debug Tool will open and close the serial port connection every time it is called; therefore, it will create extra overhead in the execution of the Arduino VI that should be taken into account. The default serial communication parameters are: 8bits, No parity, 1 stop bit. The user can change the baud rate



via the Baud control. Wire the string to send to the host via serial communication to the Value string control.



**Baud** specified the baud rate for the serial communication with the Arduino Target.

**Get Free Memory(F)**: If set to True, Debug Tool will extract the number of available bytes in RAM and append it along with Value to be written to serial port.

**Value** is the string that will be written to the serial port.

**error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**code** is the error or warning code. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**source** describes the origin of the error or warning. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**error out** passes error or warning information out of a VI to be used by other VIs. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**code** is the error or warning code. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

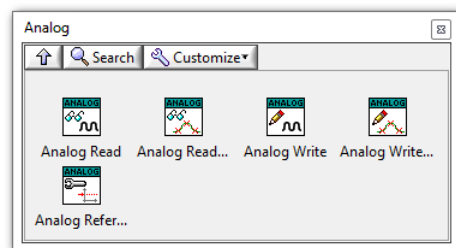
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**source** string describes the origin of the error or warning. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

## Analog Palette

**Installed With:** Arduino Compatible Compiler for LabVIEW



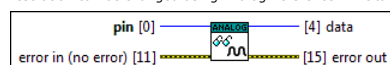
This palette contains APIs for configuring analog pins and reading and writing analog voltages. For Analog Write on a target without a DAC, this will output a pulsed width modulated (PWM) output on a digital pins. Some targets, like the Due, have a true analog output via a DAC, so the same API can be used to produce an analog voltage output. Changing the resolution of read and write is also only supported on the Due target.

©2015 TSXperts/Aledyne. All rights reserved

## Analog Read VI

**Installed With:** Arduino Compatible Compiler for LabVIEW

Reads the value from the specified analog pin in **pin**. The Arduino board contains a 6 channel (8 channels on the Mini and Nano, 16 on the Mega), 10-bit analog to digital converter. This means that it will map input voltages between 0 and 5 volts into integer values between 0 and 1023. This yields a resolution between readings of: 5 volts / 1024 units or, .0049 volts (4.9 mV) per unit. The input range and resolution can be changed using Analog Reference.vi. It takes about 100 microseconds (0.0001s) to read an analog input, so the maximum reading rate is about 10,000 times a second.



**pin** defines which analog pin on the Arduino to read data from.

**error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**code** is the error or warning code. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.


NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.


**source** describes the origin of the error or warning. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.


NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.


**data** returns the data read from the analog pin based on the full scale resolution of the pin.



 **error out** passes error or warning information out of a VI to be used by other VIs. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

 **status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

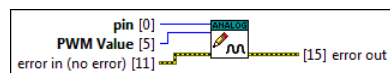
 **code** is the error or warning code. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.


 **source** string describes the origin of the error or warning. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.


## Analog Write VI


**Installed With:** Arduino Compatible Compiler for LabVIEW


Writes an analog value (PWM wave) set by **PWM Value** to a pin specified by **pin**. Can be used to light a LED at varying brightnesses or drive a motor at various speeds. After a call to Analog Write.vi, the pin will generate a steady square wave of the specified duty cycle until the next call to Analog Write.vi (or a call to Digital Read.vi or Digital Write.vi on the same pin). The frequency of the PWM signal on most pins is approximately 490 Hz. On the Arduino Uno and similar boards, pins 5 and 6 have a frequency of approximately 980 Hz. Pins 3 and 11 on the Arduino Leonardo also run at 980 Hz. On most Arduino boards (those with the ATmega168 or ATmega328), this function works on pins 3, 5, 6, 9, 10, and 11. On the Arduino Mega, it works on pins 2 - 13 and 44 - 46. Older Arduino boards with an ATmega8 only support Analog Write.vi on pins 9, 10, and 11. The Arduino Due supports Analog Write.vi on pins 2 through 13, plus pins DAC0 and DAC1. Unlike the PWM pins, DAC0 and DAC1 are Digital to Analog converters, and act as true analog outputs. You do not need to call Pin Mode.vi to set the pin as an output before calling Analog Write.vi. The Analog Write function has nothing to do with the analog pins or the Analog Read function. On the Arduino Due, there are 2 true DAC's. To access these outputs, use pin 66 for DAC0 and pin 67 for DAC1.





 **pin** defines which analog pin on the Arduino to write data to. For the Due, use pin 66 for DAC0 and pin 67 for DAC1.


 **PWM Value** defines the value to write to an analog pin using pulse width modulation.


 **error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.


 **status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.


 **code** is the error or warning code. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

 **source** describes the origin of the error or warning. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

 **error out** passes error or warning information out of a VI to be used by other VIs. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

 **status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

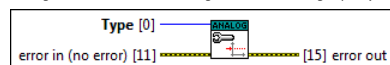
 **code** is the error or warning code. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

 **source** string describes the origin of the error or warning. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

## Analog Reference VI

**Installed With:** Arduino Compatible Compiler for LabVIEW

Configures the reference voltage used for analog input (i.e. the value used as the top of the input range). The options for **Type** are listed below.



 **Type** of reference voltage as follows:

Non-Mega Boards:


- 0: EXTERNAL - the voltage applied to the AREF pin (0 to 5V only) is used as the reference.
- 1: DEFAULT - the default analog reference of 5 volts (on 5V Arduino boards) or 3.3 volts (on 3.3V Arduino boards).
- 3: INTERNAL - a built-in reference, equal to 1.1 volts on the ATmega168 or ATmega328 and 2.56 volts on the ATmega8.


Mega Boards:


- 0: EXTERNAL - the voltage applied to the AREF pin (0 to 5V only) is used as the reference.
- 1: DEFAULT - the default analog reference of 5 volts (on 5V Arduino boards) or 3.3 volts (on 3.3V Arduino boards).
- 2: INTERNAL1V1 - a built-in 1.1V reference.
- 3: INTERNAL2V56 - a built-in 2.56V reference.


Tiny Boards:

- 0: DEFAULT - the default analog reference of 5 volts (on 5V Arduino boards) or 3.3 volts (on 3.3V Arduino boards).
- 1: EXTERNAL - the voltage applied to the AREF pin (0 to 5V only) is used as the reference.
- 2: INTERNAL - an built-in reference, equal to 1.1 volts on the ATmega168 or ATmega328 and 2.56 volts on the ATmega8.

 **error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

 **status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

 **code** is the error or warning code. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

 **source** describes the origin of the error or warning. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

information about the error.

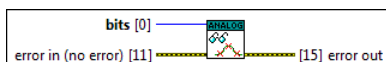
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

- error out** passes error or warning information out of a VI to be used by other VIs. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.
- status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.
- code** is the error or warning code. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.
- source** string describes the origin of the error or warning. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

## Analog Read Resolution VI

**Installed With:** Arduino Compatible Compiler for LabVIEW

Implements the analogReadResolution() Arduino function and sets the size in **bits** of the value returned by Analog Read.vi. This is only supported on the Arduino Due target. It defaults to 10 bits (returns values between 0-1023 from Analog Read.vi) for backward compatibility with AVR based boards. The Arduino Due has 12-bit ADC capabilities that can be accessed by changing the resolution to 12. This will return values from Analog Read.vi between 0 and 4095.

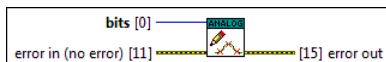


- bits** of analog data resolution.
- error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.
- status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.
- code** is the error or warning code. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.
- source** describes the origin of the error or warning. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.
- error out** passes error or warning information out of a VI to be used by other VIs. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.
- status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.
- code** is the error or warning code. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.
- source** string describes the origin of the error or warning. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

## Analog Write Resolution VI

**Installed With:** Arduino Compatible Compiler for LabVIEW

Implements the Arduino analogWriteResolution() function and sets the resolution in **bits** of the Analog Write.vi function. This is only supported on the Arduino Due target. It defaults to 8 bits (values between 0-255) for backward compatibility with AVR based boards. The Due has the following hardware capabilities: 12 pins which default to 8-bit PWM, like the AVR-based boards. These can be changed to 12-bit resolution. 2 pins with 12-bit DAC (Digital-to-Analog Converter) By setting the write resolution to 12, you can use Analog Write.vi with values between 0 and 4095 to exploit the full DAC resolution or to set the PWM signal without rolling over.

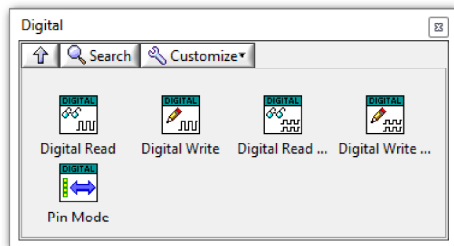


- bits** of analog data resolution.
- error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.
- status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.
- code** is the error or warning code. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.
- source** describes the origin of the error or warning. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.
- error out** passes error or warning information out of a VI to be used by other VIs. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.
- status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

- I32** **code** is the error or warning code. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.
- src** **source** string describes the origin of the error or warning. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

## Digital Palette

**Installed With:** Arduino Compatible Compiler for LabVIEW



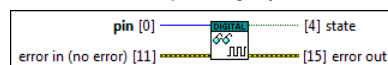
This palette contains APIs for configuring digital pins and reading and writing digital inputs and outputs. Pin Mode.vi must be used before using Digital Read or Write in order to configure the pin as an input or output. If used as an input, a pin can also be configured to use an internal pull-up resistor.

©2015 TSXperts/Aledyne. All rights reserved

## Digital Read VI

**Installed With:** Arduino Compatible Compiler for LabVIEW

Reads the value from a specified digital **pin** and returns either high (true) or low (false).

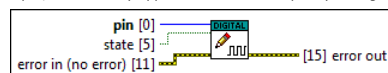


- 08** **pin** defines which digital pin on the Arduino to read from.
- err** **error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.
- TF** **status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.
- I32** **code** is the error or warning code. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.
- src** **source** describes the origin of the error or warning. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.
- TF** **state** is the state read from the digital pin (Low or High).
- err** **error out** passes error or warning information out of a VI to be used by other VIs. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.
- TF** **status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.
- I32** **code** is the error or warning code. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.
- src** **source** string describes the origin of the error or warning. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

## Digital Write VI

**Installed With:** Arduino Compatible Compiler for LabVIEW

Write a high (true) or low (false) state to a digital **pin**. If the pin has been configured as an OUTPUT with Pin Mode.vi, its voltage will be set to the corresponding value: 5V (or 3.3V on 3.3V boards) for high, 0V (ground) for low. If the pin is configured as an INPUT, Digital Write.vi will enable (high) or disable (low) the internal pullup on the input pin. It is recommended to set the Pin Mode to INPUT\_PULLUP to enable the internal pull-up resistor. See the digital pins tutorial for more information. NOTE: If you do not set the Pin Mode to OUTPUT, and connect an LED to a pin, when calling Digital Write.vi with a high input, the LED may appear dim. Without explicitly calling Pin Mode.vi, Digital Write.vi will have enabled the internal pull-up resistor, which acts like a large current-limiting resistor.



- 08** **pin** defines which digital pin on the Arduino to write to.
- TF** **state** is the state to write to the digital pin (Low or High).
- err** **error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.
- TF** **status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**code** is the error or warning code. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**source** describes the origin of the error or warning. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**error out** passes error or warning information out of a VI to be used by other VIs. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**code** is the error or warning code. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

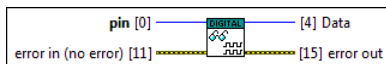
**source** string describes the origin of the error or warning. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

## Digital Read Port VI

**Installed With:** Arduino Compatible Compiler for LabVIEW

Reads the value from a specified digital port mapped to the indicated **pin** and returns the entire 8-bit value of the data on the port. Any pin on the port can be input. Refer to the Arduino documentation and schematic to determine the mapping of the pin to port. For example, if the indicated pin is wired to PORTB.2 on the Arduino, the 8-bit value of Port B will be read and returned. Note: You must ensure to set the direction of each individual pin using Pin Mode.vi before this can be used.



**pin** defines which digital pin on the Arduino to read from its mapped Port.

**error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**code** is the error or warning code. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**source** describes the origin of the error or warning. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**Data** is the 8-bit port value to write to the specified pin's port. Only the data starting from **Start Pin** will be updated.

**error out** passes error or warning information out of a VI to be used by other VIs. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**code** is the error or warning code. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

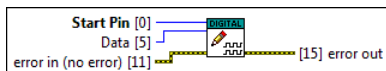
**source** string describes the origin of the error or warning. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

## Digital Write Port VI

**Installed With:** Arduino Compatible Compiler for LabVIEW

Writes **Data** to a specified digital port mapped to the indicated **Start Pin**. Only pins after the start pin will be updated. In this way, you can maintain data on pins of the same data port before the start pin. Refer to the Arduino documentation and schematic to determine the mapping of the pin to port. For example, if the **Start Pin** specified is wired to PORTB.2 on the Arduino, only bits 2-8 will be updated from the input <Data>. Bit 1 will retain its current value. Note: You must ensure to set the direction of each individual pin using Pin Mode.vi before this can be used.



**Start Pin** defines which digital pin on the Arduino to start updating values to.

**Data** is the 8-bit port value to write to the specified pin's port. Only the data starting from **Start Pin** will be updated.

**error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.





NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**code** is the error or warning code. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**source** describes the origin of the error or warning. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

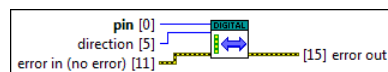
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.











-  **error out** passes error or warning information out of a VI to be used by other VIs. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.
-  **status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.
-  **code** is the error or warning code. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.
-  **source** string describes the origin of the error or warning. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

## Pin Mode VI

**Installed With:** Arduino Compatible Compiler for LabVIEW

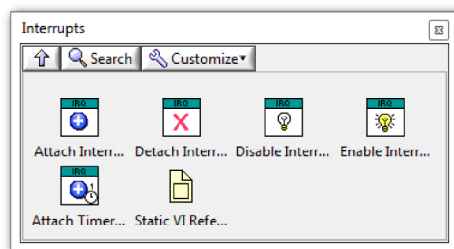
Configures the specified **pin** to behave either as an input or an output using **direction**. See the Arduino description of digital pins for details on the functionality of the pins. As of Arduino 1.0.1, it is possible to enable the internal pullup resistors with the mode INPUT\_PULLUP. Additionally, the INPUT mode explicitly disables the internal pullups.



-  **pin** defines which digital pin on the Arduino to configure.
-  **direction** defines the data direction of the digital pin selected, input, output, or internal pullup.
-  **error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.
-  **status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.
-  **code** is the error or warning code. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.
-  **source** describes the origin of the error or warning. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.
-  **error out** passes error or warning information out of a VI to be used by other VIs. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.
-  **status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.
-  **code** is the error or warning code. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.
-  **source** string describes the origin of the error or warning. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

## Interrupts Palette

**Installed With:** Arduino Compatible Compiler for LabVIEW



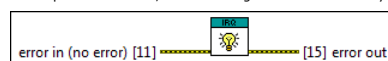
The Interrupts palette provides a way to enable and disable Arduino interrupts. It is possible to link a VI as a callback to when a specified Interrupt is triggered. In order to share data between a main VI and a callback (Interrupt) VI, global variables must be used. Please refer to the examples to demonstrate how to do this. For example, an interrupt can be enabled for a digital input pin transition and the even can trigger running of a callback VI.

©2015 TSXperts/Aledyne. All rights reserved

## Enable Interrupts VI

**Installed With:** Arduino Compatible Compiler for LabVIEW

Re-enables interrupts (after they've been disabled by Disable Interrupts). Interrupts allow certain important tasks to happen in the background and are enabled by default. Some functions will not work while interrupts are disabled, and incoming communication may be ignored. Interrupts can slightly disrupt the timing of code, however, and may be disabled for particularly critical sections of code.



- error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.
- status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.
- code** is the error or warning code. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.
- source** describes the origin of the error or warning. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.
- error out** passes error or warning information out of a VI to be used by other VIs. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.
- status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.
- code** is the error or warning code. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.
- source** string describes the origin of the error or warning. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

## Disable Interrupts VI

**Installed With:** Arduino Compatible Compiler for LabVIEW

Disables interrupts (you can re-enable them with Enable Interrupts). Interrupts allow certain important tasks to happen in the background and are enabled by default. Some functions will not work while interrupts are disabled, and incoming communication may be ignored. Interrupts can slightly disrupt the timing of code, however, and may be disabled for particularly critical sections of code.

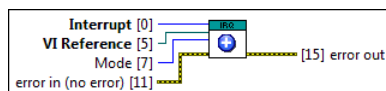


- error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.
- status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.
- code** is the error or warning code. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.
- source** describes the origin of the error or warning. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.
- error out** passes error or warning information out of a VI to be used by other VIs. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.
- status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.
- code** is the error or warning code. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.
- source** string describes the origin of the error or warning. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

## Attach Interrupt VI

**Installed With:** Arduino Compatible Compiler for LabVIEW

Specifies a callback VI to call in **VI Reference** when the specified **Interrupt** occurs. **Mode** specifies the pin transition mode that causes the interrupt to occur. Replaces any previous VI that was attached to the interrupt. Most Arduino boards have two external interrupts: numbers 0 (on digital pin 2) and 1 (on digital pin 3). Refer to table on front panel for interrupt numbers for each Arduino board. Also refer to <http://arduino.cc/en/Reference/AttachInterrupt> for the most recent updates for supported boards and interrupts. The Arduino Due board has powerful interrupt capabilities that allows you to attach an interrupt function on all available pins. You can directly specify the pin number in Attach Interrupt.vi. Note Inside the attached VI, any Delay calls won't work and the value returned by Tick Count vis will not increment. Serial data received while in the function may be lost. To transfer data in and out of the callback VI, global variables must be used.



- Interrupt** defines the interrupt number of the processor.
- VI Reference** is the reference to the callback VI that will run when the interrupt occurs.
- Mode** defines when the interrupt should be triggered. Options are defined as follows: LOW: to trigger the interrupt whenever the pin is low. CHANGE: to trigger the interrupt whenever the pin changes value. RISING: to trigger when the pin goes from low to high. FALLING: for when the pin goes from high to low. The Due board allows also: HIGH to trigger the interrupt whenever the pin is high (Arduino Due only).
- error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.
- status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.



help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**I321** **code** is the error or warning code. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**I324** **source** describes the origin of the error or warning. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**I321** **error out** passes error or warning information out of a VI to be used by other VIs. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**I324** **status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**I321** **code** is the error or warning code. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

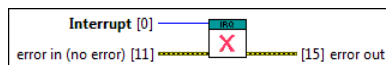
**I324** **source** string describes the origin of the error or warning. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

## Detach Interrupt VI

**Installed With:** Arduino Compatible Compiler for LabVIEW

Turns off the given interrupt.



**I321** **Interrupt** defines the interrupt number of the processor.

**I321** **error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**I324** **status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**I321** **code** is the error or warning code. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**I324** **source** describes the origin of the error or warning. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**I321** **error out** passes error or warning information out of a VI to be used by other VIs. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**I324** **status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**I321** **code** is the error or warning code. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

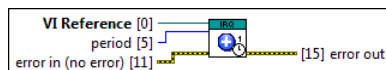
**I324** **source** string describes the origin of the error or warning. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

## Attach Timer1 Interrupt VI

**Installed With:** Arduino Compatible Compiler for LabVIEW

Enables Timer1 and specifies a callback VI to call in **VI Reference** when the timer interrupt occurs. The **period** specifies how often to trigger the callback in microseconds. This interrupt will only compile on AVR platforms and will not work on the Arduino Due. Note: Inside the attached VI, any Delay calls won't work and the value returned by Tick Count vis will not increment. Serial data received while in the function may be lost. To transfer data in and out of the callback VI, global variables must be used. Also, the referenced callback VI cannot have any input or outputs wired to the terminals.



**I321** **VI Reference** is the reference to the callback VI that will run when the interrupt occurs.

**I321** **period** in microseconds to trigger the callback VI.

**I321** **error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**I324** **status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

**I321** **code** is the error or warning code. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

**I324** **source** describes the origin of the error or warning. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

**I321** **error out** passes error or warning information out of a VI to be used by other VIs. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

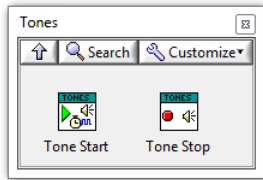
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**I324** **status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

- I32** **code** is the error or warning code. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.
- 13c** **source** string describes the origin of the error or warning. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

## Tones Palette

**Installed With:** Arduino Compatible Compiler for LabVIEW



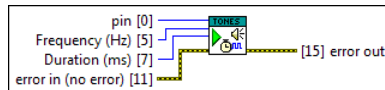
The Tones palette contains Arduino specific APIs in order to start and stop tones on an Arduino target via a digital output pin. These are used to generate a square wave of the specified Frequency (Hz) (and 50% duty cycle) on a digital pin. This functionality is currently not supported on the Arduino Due. Use of the Tone Start.vi function will interfere with PWM output on pins 3 and 11 (on boards other than the Mega). It is not possible to generate tones lower than 31Hz.

©2015 TSXperts/Aledyne. All rights reserved

## Tone Start VI

**Installed With:** Arduino Compatible Compiler for LabVIEW

Generates a square wave of the specified **Frequency (Hz)** (and 50% duty cycle) on a **pin**. A **Duration (ms)** in milliseconds can be specified, otherwise if unconnected or set to 0, the wave continues until a call to Tone Stop.vi. The pin can be connected to a piezo buzzer or other speaker to play tones. Only one tone can be generated at a time. If a tone is already playing on a different pin, the call to Tone Start.vi will have no effect. If the tone is playing on the same pin, the call will set its frequency. Use of the Tone Start.vi function will interfere with PWM output on pins 3 and 11 (on boards other than the Mega). It is not possible to generate tones lower than 31Hz. This functionality is currently not supported on the Arduino Due. NOTE: if you want to play different pitches on multiple pins, you need to call Tone Stop.vi on one pin before calling Tone Start.vi on the next pin.

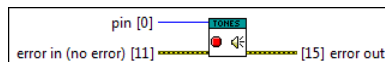


- U8** **pin** defines which digital pin to use for tone generation.
- U16** **Frequency (Hz)** defines the frequency of the tone to emit on the digital pin.
- U32** **Duration (ms)** defines the duration of the tone in milliseconds. If set to 0, the tone runs indefinitely or until Tone Stop.vi is called.
- E32** **error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.
- TF** **status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.
- I32** **code** is the error or warning code. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.
- 13c** **source** describes the origin of the error or warning. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.
- E32** **error out** passes error or warning information out of a VI to be used by other VIs. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.
- TF** **status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.
- I32** **code** is the error or warning code. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.
- 13c** **source** string describes the origin of the error or warning. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

## Tone Stop VI

**Installed With:** Arduino Compatible Compiler for LabVIEW

Stops the generation of a square wave triggered by Tone Start.vi on the specified **pin**. Has no effect if no tone is being generated. This functionality is currently not supported on the Arduino Due. NOTE: if you want to play different pitches on multiple pins, you need to call Tone Stop.vi on one pin before calling Tone Start.vi on the next pin.



- U8** **pin** defines which digital pin to use for tone generation.
- E32** **error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.
- TF** **status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.
- I32** **code** is the error or warning code. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.



error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**134** **source** describes the origin of the error or warning. Right-click the **error in control** on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**135** **error out** passes error or warning information out of a VI to be used by other VIs. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**136** **status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**137** **code** is the error or warning code. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

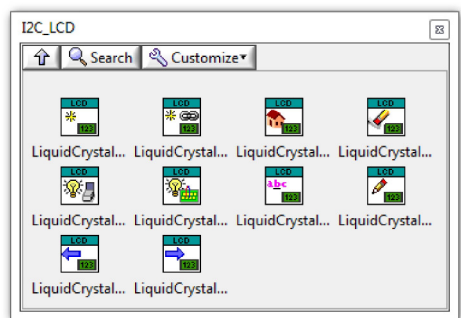
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**138** **source** string describes the origin of the error or warning. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

## I2C LCD Palette

**Installed With:** Arduino Compatible Compiler for LabVIEW



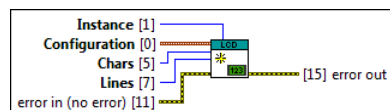
The I2C LCD Palette includes APIs for writing data to a Sainsmart (or equivalent) LCD with an I2C LCD Controller. An Arduino I2C port using 2 digital pins must be used to interface to this palette.

©2015 TSXperts/Aledyne. All rights reserved

## LiquidCrystal\_I2C VI

**Installed With:** Arduino Compatible Compiler for LabVIEW

The **Instance** is a reference to the instance of the LCD class. If only one LCD is used then this should be set to 0. Creates an instance of the LiquidCrystalI2C Class. This class is specific to the sainsmart LCD2004 controller. **Addr:** The I2C Address to be used (usually 0x3F or decimal 63 for the Sainsmart controller). **ENPin:** The number of the sainsmart controller pin that is connected to the enable pin on the LCD. **RWPIn:** The number of the sainsmart controller pin that is connected to the RW pin on the LCD. **RSPin:** The number of the sainsmart controller pin that is connected to the RS pin on the LCD. **D4Pin, D5Pin, D6Pin, D7Pin:** The numbers of the sainsmart controller pins that are connected to the corresponding data pins on the LCD. The LCD can only be controlled using 4-bit mode through the four data lines (d4, d5, d6, d7). **Chars** specifies the number of characters in a line of the LCD. **Lines** specifies how many lines are in the LCD.



**139** **Instance** is a reference to the instance of the LCD class.

**140** **Configuration** is the configuration of the I2C LCD controller.

**141** **Addr** is the I2C Address to be used (usually 0x3F or decimal 63 for the Sainsmart controller).

**142** **ENPin** is the number of the controller pin that is connected to the enable pin on the LCD.

**143** **RWPIn** is the number of the controller pin that is connected to the RW pin on the LCD.

**144** **RSPin** is the number of the controller pin that is connected to the RS pin on the LCD.

**145** **D4Pin** is the controller pin that is connected to the corresponding data pins on the LCD. The LCD can only be controlled using 4-bit mode through the four data lines (d4, d5, d6, d7).

**146** **D5Pin** is the controller pin that is connected to the corresponding data pins on the LCD. The LCD can only be controlled using 4-bit mode through the four data lines (d4, d5, d6, d7).

**147** **D6Pin** is the controller pin that is connected to the corresponding data pins on the LCD. The LCD can only be controlled using 4-bit mode through the four data lines (d4, d5, d6, d7).

**148** **D7Pin** is the controller pin that is connected to the corresponding data pins on the LCD. The LCD can only be controlled using 4-bit mode through the four data lines (d4, d5, d6, d7).

**149** **Chars** defines how many characters or columns are on the LCD.

**150** **Lines** defines how many lines or rows are on the LCD.

**151** **error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**152** **status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**153** **code** is the error or warning code. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**source** describes the origin of the error or warning. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**error out** passes error or warning information out of a VI to be used by other VIs. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

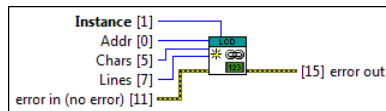
**code** is the error or warning code. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**source** string describes the origin of the error or warning. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

## LiquidCrystal\_I2C Express VI

**Installed With:** Arduino Compatible Compiler for LabVIEW

The **Instance** is a reference to the instance of the LCD class. If only one LCD is used then this should be set to 0. Creates an instance of the LiquidCrystalI2C Class. Also wraps all initialization by configuring the LCD data and backlight pins to the standard configuration for the LCD2004 controller. The Enable Pin is set to 2, RW Pin is set to 1, RSPin is set to 0, and D4-D7 is set to pins 4-7 on the LCD controller. Also, the backlight pin is set to 3 and turned on. **Addr:** The I2C Address to be used (usually 0x3F or decimal 63 for the Sainsmart controller). **Chars** specifies the number of characters in a line of the LCD. **Lines** specifies how many lines are in the LCD.



**Instance** is a reference to the instance of the LCD class.

**Addr:** The I2C Address to be used (usually 0x3F or decimal 63 for the Sainsmart controller).

**Chars** defines how many characters or columns are on the LCD.

**Lines** defines how many lines or rows are on the LCD.

**error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

**status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

**code** is the error or warning code. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

**source** describes the origin of the error or warning. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

**error out** passes error or warning information out of a VI to be used by other VIs. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

**status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

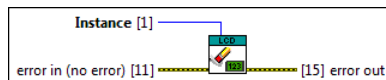
**code** is the error or warning code. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

**source** string describes the origin of the error or warning. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

## LiquidCrystal\_I2C clear VI

**Installed With:** Arduino Compatible Compiler for LabVIEW

The **Instance** is a reference to the instance of the LCD class. If only one LCD is used then this should be set to 0. This VI clears the LCD screen and positions the cursor in the upper-left corner.



**Instance** is a reference to the instance of the LCD class.

**error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**code** is the error or warning code. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**source** describes the origin of the error or warning. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**error out** passes error or warning information out of a VI to be used by other VIs. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**code** is the error or warning code. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

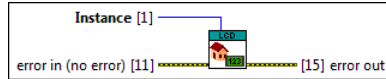
**source** string describes the origin of the error or warning. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

## LiquidCrystal\_I2C home VI

**Installed With:** Arduino Compatible Compiler for LabVIEW

The **Instance** is a reference to the instance of the LCD class. If only one LCD is used then this should be set to 0. Positions the cursor in the upper-left of the LCD. That is, use that location in outputting subsequent text to the display. To also clear the display, use LiquidCrystal\_I2C clear.vi instead.



**Instance** is a reference to the instance of the LCD class.

**error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**code** is the error or warning code. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**source** describes the origin of the error or warning. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**error out** passes error or warning information out of a VI to be used by other VIs. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**code** is the error or warning code. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

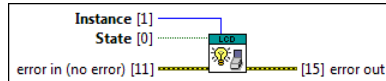
**source** string describes the origin of the error or warning. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

## LiquidCrystal\_I2C set backlight VI

**Installed With:** Arduino Compatible Compiler for LabVIEW

The **Instance** is a reference to the instance of the LCD class. If only one LCD is used then this should be set to 0. Turns the backlight of the LCD on or off as specified by **State**.



**Instance** is a reference to the instance of the LCD class.

**State** sets the on/off state of the backlight.

**error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**code** is the error or warning code. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**source** describes the origin of the error or warning. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**error out** passes error or warning information out of a VI to be used by other VIs. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**code** is the error or warning code. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

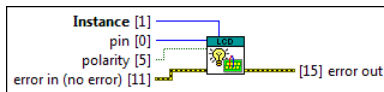
**source** string describes the origin of the error or warning. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

## LiquidCrystal\_I2C set backlight pin VI

**Installed With:** Arduino Compatible Compiler for LabVIEW

The **Instance** is a reference to the instance of the LCD class. If only one LCD is used then this should be set to 0. Sets the appropriate pin and polarity of the backlight control pin of the sainsmart controller. By default, **pin** should be set to 3 and **polarity** set to false to drive the backlight correctly.



**Instance** is a reference to the instance of the LCD class.

**pin** defines which pin the backlight is wired to on the LCD module.

**polarity** defines the polarity of the backlight pin on the LCD module. This is usually false for the Sainsmart I2C LCD modules.

**error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**code** is the error or warning code. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**source** describes the origin of the error or warning. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**error out** passes error or warning information out of a VI to be used by other VIs. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

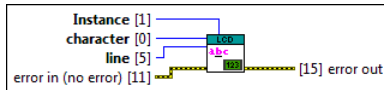
**code** is the error or warning code. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**source** string describes the origin of the error or warning. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

## LiquidCrystal\_I2C set cursor VI

**Installed With:** Arduino Compatible Compiler for LabVIEW

The **Instance** is a reference to the instance of the LCD class. If only one LCD is used then this should be set to 0. Position the LCD cursor; that is, set the location at which subsequent text written to the LCD will be displayed. Text will be placed at the specified **character** and **line**.



**Instance** is a reference to the instance of the LCD class.

**character** offset of where to place text at on the LCD.

**line** where to place text at on the LCD.

**error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**code** is the error or warning code. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**source** describes the origin of the error or warning. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**error out** passes error or warning information out of a VI to be used by other VIs. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

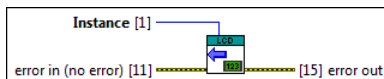
**code** is the error or warning code. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**source** string describes the origin of the error or warning. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

## LiquidCrystal\_I2C scroll left VI

**Installed With:** Arduino Compatible Compiler for LabVIEW





The **Instance** is a reference to the instance of the LCD class. If only one LCD is used then this should be set to 0. Scrolls the contents of the display (text and cursor) one space to the left.









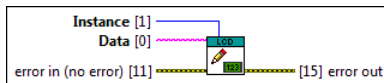
-  shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.
-  **status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.
-  **code** is the error or warning code. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.
-  **source** string describes the origin of the error or warning. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**source** string describes the origin of the error or warning. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.







**Instance** is a reference to the instance of the LCD class.

**Data** to write to the LCD.

**error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

**code** is the error or warning code. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

**source** describes the origin of the error or warning. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**error out** passes error or warning information out of a VI to be used by other VIs. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

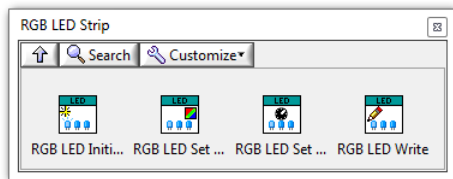
**status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

**code** is the error or warning code. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

**source** string describes the origin of the error or warning. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

## RGB LED Palette

Installed With: Arduino Compatible Compiler for LabVIEW



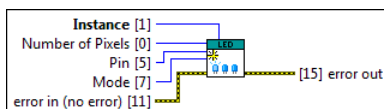
The RGB LED Palette includes APIs for writing data to 1-wire WS2811 and WS2812 controllers on RGB LED strips similar to the ones found [here](#). A single Arduino digital output pin must be used to interface to this palette.

©2015 TSXperts/Aledyne. All rights reserved

## RGB LED Initialize VI

Installed With: Arduino Compatible Compiler for LabVIEW

The **Instance** is a reference to the instance of the strip LED class. If only one LED strip is used then this should be set to 0. Up to 3 LED strips can be created and addressed on different pins in the same application using a different instance number. This is specific to the WS2811 and WS2812 controllers.



**Instance** is a reference to the instance of the RGB LED class.

**Number of Pixels** specifies how many pixels or LEDs exist on the strip or how many you want to control. This can be set to any number less than the number of LEDs on the strip(s) if you don't need all of them to illuminate.

**Pin** specifies which pin on the Arduino that the Data pin on the LED strip is connected to.

**Mode** specifies the frequency and the RGB data packing order of the LED strip. Check your LED strip documentation for details. For the WS2812 controller, this is 800kHz. For most NeoPixel products the wiring is GRB. For v1 FLORA pixels with the WS2811 driver, it is usually 400kHz RGB.

**error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

**code** is the error or warning code. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

**source** describes the origin of the error or warning. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**error out** passes error or warning information out of a VI to be used by other VIs. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**TF** **status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**I32** **code** is the error or warning code. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

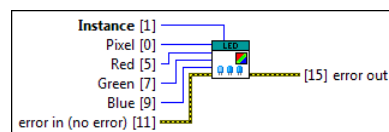
**abc** **source** string describes the origin of the error or warning. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

## RGB LED Set Pixel Color VI

**Installed With:** Arduino Compatible Compiler for LabVIEW

Sets the pixel (LED) color of a single LED on the strip using an 8-bit representation for **Red**, **Green** and **Blue**. This does not actually update the LED, this just stores the new color setting until LED Write.vi is called.



**UB** **Instance** is a reference to the instance of the RGB LED class.

**UI6** **Pixel** specifies which pixel (or LED) to update.

**UB** **Red** specifies the amount of red color (0-255) to output to the specified **Pixel** or LED.

**UB** **Green** specifies the amount of green color (0-255) to output to the specified **Pixel** or LED.

**UB** **Blue** specifies the amount of blue color (0-255) to output to the specified **Pixel** or LED.

**E2** **error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**TF** **status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**I32** **code** is the error or warning code. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**abc** **source** describes the origin of the error or warning. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**E2** **error out** passes error or warning information out of a VI to be used by other VIs. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**TF** **status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**I32** **code** is the error or warning code. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

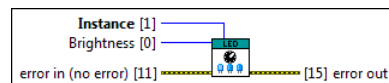
**abc** **source** string describes the origin of the error or warning. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

## RGB LED Set Brightness VI

**Installed With:** Arduino Compatible Compiler for LabVIEW

Sets the LED brightness on a scale of 0-255 on the entire strip.



**UB** **Instance** is a reference to the instance of the RGB LED class.

**UB** **Brightness** sets the LED brightness on a scale of 0-255 on the entire strip.

**E2** **error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**TF** **status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**I32** **code** is the error or warning code. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**abc** **source** describes the origin of the error or warning. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**E2** **error out** passes error or warning information out of a VI to be used by other VIs. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

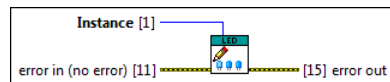
**code** is the error or warning code. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**source** string describes the origin of the error or warning. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

## RGB LED Write VI

**Installed With:** Arduino Compatible Compiler for LabVIEW

Writes all outstanding pixel color and brightness setting changes to the LEDs since last written.



**Instance** is a reference to the instance of the RGB LED class.

**error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**code** is the error or warning code. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**source** describes the origin of the error or warning. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**error out** passes error or warning information out of a VI to be used by other VIs. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

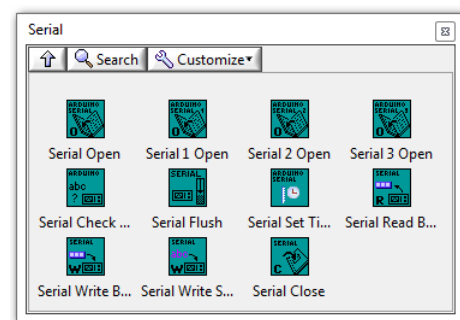
**status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**code** is the error or warning code. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**source** string describes the origin of the error or warning. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

## Serial Palette

**Installed With:** Arduino Compatible Compiler for LabVIEW



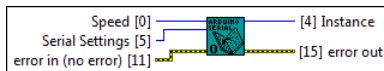
The serial palette is used to interface with the Serial port(s) on the Arduino target. For most boards, only one serial port is available. For those, the "Serial Open.vi" must be used. A small number of boards, like the Mega, support up to 4 serial ports. The second serial port is actually labeled "Serial 1" and the third is labeled "Serial 2" and so forth. Therefore, use the corresponding "Serial X Open.vi" to interface with those serial ports. Using any serial primitive outside this palette will result in a compilation error.

©2015 TSXperts/Aledyne. All rights reserved

## Serial Open VI

**Installed With:** Arduino Compatible Compiler for LabVIEW

Initializes serial communication on Serial Port 1 of the Arduino board. All Arduino boards have at least one serial port (also known as a UART or USART): Serial. It communicates on digital pins 0 (RX) and 1 (TX) as well as with the computer via USB. Thus, if you use these functions, you cannot also use pins 0 and 1 for digital input or output. The Arduino Mega has three additional serial ports: Serial 1 on pins 19 (RX) and 18 (TX), Serial 2 on pins 17 (RX) and 16 (TX), Serial 3 on pins 15 (RX) and 14 (TX). The Arduino Due has three additional 3.3V TTL serial ports: Serial 1 on pins 19 (RX) and 18 (TX); Serial 2 on pins 17 (RX) and 16 (TX), Serial 3 on pins 15 (RX) and 14 (TX). Pins 0 and 1 are also connected to the corresponding pins of the ATmega16U2 USB-to-TTL Serial chip, which is connected to the USB debug port. Additionally, there is a native USB-serial port on the SAM3X chip, SerialUSB. This VI also sets the data rate in bits per second (baud) for serial data transmission. For communicating with the computer, use one of these rates: 300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600, or 115200. You can, however, specify other rates. An optional second argument configures the data, parity, and stop bits. The default is 8 data bits, no parity, one stop bit.



**Speed** is the baud rate of the serial port.

**Serial Settings** defines the data width, parity, and stop bits of the serial port.

**error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**code** is the error or warning code. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**source** describes the origin of the error or warning. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**Instance** is the reference to the serial port. This should be passed into subsequent serial port VIs. This can also be hardcoded for each subsequent serial access as follows - Serial: 0, Serial 1: 1, Serial 2: 2, and Serial3: 3).

**error out** passes error or warning information out of a VI to be used by other VIs. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**code** is the error or warning code. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

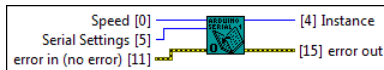
**source** string describes the origin of the error or warning. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

## Serial 1 Open VI

**Installed With:** Arduino Compatible Compiler for LabVIEW

Initializes serial communication on Serial Port 2 of the Arduino board. All Arduino boards have at least one serial port (also known as a UART or USART): Serial. It communicates on digital pins 0 (RX) and 1 (TX) as well as with the computer via USB. Thus, if you use these functions, you cannot also use pins 0 and 1 for digital input or output. The Arduino Mega has three additional serial ports: Serial 1 on pins 19 (RX) and 18 (TX), Serial 2 on pins 17 (RX) and 16 (TX), Serial 3 on pins 15 (RX) and 14 (TX). The Arduino Due has three additional 3.3V TTL serial ports: Serial 1 on pins 19 (RX) and 18 (TX); Serial 2 on pins 17 (RX) and 16 (TX), Serial 3 on pins 15 (RX) and 14 (TX). Pins 0 and 1 are also connected to the corresponding pins of the ATmega16U2 USB-to-TTL Serial chip, which is connected to the USB debug port. Additionally, there is a native USB-serial port on the SAM3X chip, SerialUSB\*. This VI also sets the data rate in bits per second (baud) for serial data transmission. For communicating with the computer, use one of these rates: 300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600, or 115200. You can, however, specify other rates. An optional second argument configures the data, parity, and stop bits. The default is 8 data bits, no parity, one stop bit.



**Speed** is the baud rate of the serial port.

**Serial Settings** defines the data width, parity, and stop bits of the serial port.

**error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**code** is the error or warning code. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**source** describes the origin of the error or warning. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**Instance** is the reference to the serial port. This should be passed into subsequent serial port VIs. This can also be hardcoded for each subsequent serial access as follows - Serial: 0, Serial 1: 1, Serial 2: 2, and Serial3: 3).

**error out** passes error or warning information out of a VI to be used by other VIs. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**code** is the error or warning code. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**source** string describes the origin of the error or warning. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

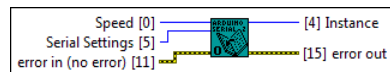
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

## Serial 2 Open VI

**Installed With:** Arduino Compatible Compiler for LabVIEW

Initializes serial communication on Serial Port 3 of the Arduino board. All Arduino boards have at least one serial port (also known as a UART or USART): Serial. It communicates on digital pins 0 (RX) and 1 (TX) as well as with the computer via USB. Thus, if you use these functions, you cannot also use pins 0 and 1 for digital input or output. The Arduino Mega has three additional serial ports: Serial 1 on pins 19 (RX) and 18 (TX), Serial 2 on pins 17 (RX) and 16 (TX), Serial 3 on pins 15 (RX) and 14 (TX). The Arduino Due has three additional 3.3V TTL serial ports: Serial 1 on pins 19 (RX) and 18 (TX); Serial 2 on pins 17 (RX) and 16 (TX), Serial 3 on pins 15 (RX) and 14 (TX). Pins 0 and 1 are also connected to the corresponding pins of the ATmega16U2 USB-to-TTL Serial chip, which is connected to the USB debug

port. Additionally, there is a native USB-serial port on the SAM3X chip, SerialUSB<sup>1</sup>. This VI also sets the data rate in bits per second (baud) for serial data transmission. For communicating with the computer, use one of these rates: 300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600, or 115200. You can, however, specify other rates. An optional second argument configures the data, parity, and stop bits. The default is 8 data bits, no parity, one stop bit.



**Speed** is the baud rate of the serial port.

**Serial Settings** defines the data width, parity, and stop bits of the serial port.

**error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**code** is the error or warning code. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**source** describes the origin of the error or warning. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**Instance** is the reference to the serial port. This should be passed into subsequent serial port VIs. This can also be hardcoded for each subsequent serial access as follows - Serial: 0, Serial 1: 1, Serial 2: 2, and Serial3: 3).

**error out** passes error or warning information out of a VI to be used by other VIs. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**code** is the error or warning code. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

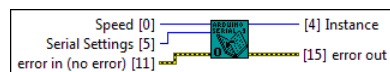
**source** string describes the origin of the error or warning. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

## Serial 3 Open VI

**Installed With:** Arduino Compatible Compiler for LabVIEW

Initializes serial communication on Serial Port 4 of the Arduino board. All Arduino boards have at least one serial port (also known as a UART or USART): Serial. It communicates on digital pins 0 (RX) and 1 (TX) as well as with the computer via USB. Thus, if you use these functions, you cannot also use pins 0 and 1 for digital input or output. The Arduino Mega has three additional serial ports: Serial 1 on pins 19 (RX) and 18 (TX), Serial 2 on pins 17 (RX) and 16 (TX), Serial 3 on pins 15 (RX) and 14 (TX). The Arduino Due has three additional 3.3V TTL serial ports: Serial 1 on pins 19 (RX) and 18 (TX); Serial 2 on pins 17 (RX) and 16 (TX), Serial 3 on pins 15 (RX) and 14 (TX). Pins 0 and 1 are also connected to the corresponding pins of the ATmega16U2 USB-to-TTL Serial chip, which is connected to the USB debug port. Additionally, there is a native USB-serial port on the SAM3X chip, SerialUSB<sup>1</sup>. This VI also sets the data rate in bits per second (baud) for serial data transmission. For communicating with the computer, use one of these rates: 300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600, or 115200. You can, however, specify other rates. An optional second argument configures the data, parity, and stop bits. The default is 8 data bits, no parity, one stop bit.



**Speed** is the baud rate of the serial port.

**Serial Settings** defines the data width, parity, and stop bits of the serial port.

**error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**code** is the error or warning code. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**source** describes the origin of the error or warning. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**Instance** is the reference to the serial port. This should be passed into subsequent serial port VIs. This can also be hardcoded for each subsequent serial access as follows - Serial: 0, Serial 1: 1, Serial 2: 2, and Serial3: 3).

**error out** passes error or warning information out of a VI to be used by other VIs. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**code** is the error or warning code. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**source** string describes the origin of the error or warning. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

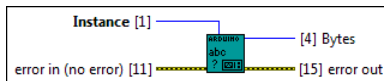
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

## Serial Check for Bytes VI

**Installed With:** Arduino Compatible Compiler for LabVIEW

Get the number of bytes (characters) available for reading from the serial port. This is data that's already arrived and stored in the serial receive buffer (which holds 64 bytes).





**Instance** is the reference to the serial port.

**error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**code** is the error or warning code. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**source** describes the origin of the error or warning. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**Bytes** available at the serial port.

**error out** passes error or warning information out of a VI to be used by other VIs. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**code** is the error or warning code. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

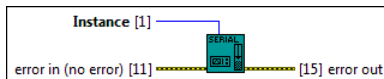
**source** string describes the origin of the error or warning. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

## Serial Flush VI

**Installed With:** Arduino Compatible Compiler for LabVIEW

Waits for the transmission of outgoing serial data to complete. (Prior to Arduino 1.0, this instead removed any buffered incoming serial data.)



**Instance** is the reference to the serial port.

**error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**code** is the error or warning code. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**source** describes the origin of the error or warning. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**error out** passes error or warning information out of a VI to be used by other VIs. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**code** is the error or warning code. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

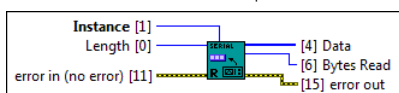
**source** string describes the origin of the error or warning. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

## Serial Read Bytes VI

**Installed With:** Arduino Compatible Compiler for LabVIEW

Serial Read Bytes.vi reads characters from the serial port into a buffer. The function terminates if the determined length has been read, or it times out (based on the setting of Serial Set Timeout.vi). This VI also returns the number of characters placed in the buffer. A 0 in **bytes read** means no valid data was found.



**Instance** is the reference to the serial port.

**Length** is the length of data to read from the serial port.

**error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**TF** **status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**I32** **code** is the error or warning code. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**ABC** **source** describes the origin of the error or warning. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**U8** **Data** is a byte array of data read from the serial port. This can be converted to a string if the data being received is ASCII by using the Byte Array to String LabVIEW primitive.

**I32** **Bytes Read** is the actual length of data read from the serial port.

**E** **error out** passes error or warning information out of a VI to be used by other VIs. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**TF** **status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**I32** **code** is the error or warning code. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

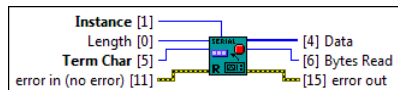
**ABC** **source** string describes the origin of the error or warning. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

## Serial Read Bytes Until VI

**Installed With:** Arduino Compatible Compiler for LabVIEW

Serial Read Bytes Until.vi reads characters from the serial port into a buffer until the terminator character is detected, the determined length has been read, or it times out (based on the setting of Serial Set Timeout.vi). This VI also returns the number of characters placed in the buffer. A 0 in **bytes read** means no valid data was found.



**U8** **Instance** is the reference to the serial port.

**I32** **Length** is the length of data to read from the serial port.

**U8** **Term Char** defines the character to stop reading when it is received.

**E** **error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**TF** **status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**I32** **code** is the error or warning code. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**ABC** **source** describes the origin of the error or warning. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**U8** **Data** is a byte array of data read from the serial port. This can be converted to a string if the data being received is ASCII by using the Byte Array to String LabVIEW primitive.

**I32** **Bytes Read** is the actual length of data read from the serial port.

**E** **error out** passes error or warning information out of a VI to be used by other VIs. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**TF** **status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**I32** **code** is the error or warning code. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

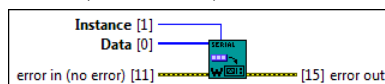
**ABC** **source** string describes the origin of the error or warning. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

## Serial Write Bytes VI

**Installed With:** Arduino Compatible Compiler for LabVIEW

Writes binary data to the serial port. This data is sent as a byte or series of bytes.



**U8** **Instance** is the reference to the serial port.

**U8** **Data** is the array of bytes to write to the serial port.

**E** **error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control





**abc** information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**error out** passes error or warning information out of a VI to be used by other VIs. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**TF** **status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**132** **code** is the error or warning code. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

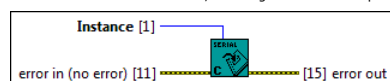
**abc** **source** string describes the origin of the error or warning. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

## Serial Close VI

**Installed With:** Arduino Compatible Compiler for LabVIEW

Disables serial communication, allowing the RX and TX pins to be used for general input and output. To re-enable serial communication, call Serial X Open.vi



**U0** **Instance** is the reference to the serial port.

**error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**TF** **status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**132** **code** is the error or warning code. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**abc** **source** describes the origin of the error or warning. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**error out** passes error or warning information out of a VI to be used by other VIs. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**TF** **status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**132** **code** is the error or warning code. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

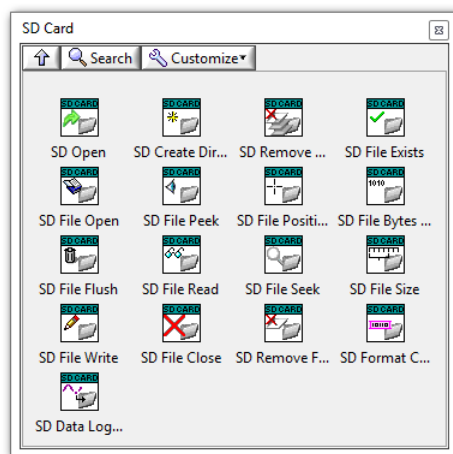
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**abc** **source** string describes the origin of the error or warning. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

## SD Card Palette

**Installed With:** Arduino Compatible Compiler for LabVIEW

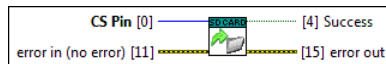


The SD Card palette includes APIs for interfacing to a file system on an SD card for creating, reading, and writing files and directories. An Arduino SPI port with dedicated Chip Select must be used to interface to this palette.

## SD Open VI

**Installed With:** Arduino Compatible Compiler for LabVIEW

Opens a reference to the SD card and initializes it.



**CS Pin** is the SPI Chip Select pin. Refer to the Arduino documentation on SPI at <http://arduino.cc/en/Reference/SDBegin>. Make sure to match the CS Pin that is appropriate for your combination of Arduino target board as well as SD Card shield.

**error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**code** is the error or warning code. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**source** describes the origin of the error or warning. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**success** indicates whether or not the operation was successful.

**error out** passes error or warning information out of a VI to be used by other VIs. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

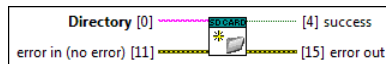
**code** is the error or warning code. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**source** string describes the origin of the error or warning. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

## SD Create Directory VI

**Installed With:** Arduino Compatible Compiler for LabVIEW

Create a directory on the SD card. This will also create any intermediate directories that don't already exist; for example, a/b/c in the Directory input will create a, b, and c. Note: The SD library uses 8.3 filenames which have at most 8 characters for files and directories optionally followed by a period and a filename extension of at most three characters. File and directory names are uppercase, although systems that use the 8.3 standard are usually case-insensitive.



**Directory** holds the directory name for the directory that will be created in the SD Card. Limited to 8 characters for each directory.

**error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**code** is the error or warning code. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**source** describes the origin of the error or warning. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**success** returns whether or not the operation was successful.

**error out** passes error or warning information out of a VI to be used by other VIs. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**code** is the error or warning code. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**source** string describes the origin of the error or warning. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

## SD Remove Directory VI

**Installed With:** Arduino Compatible Compiler for LabVIEW

Removes the input **Directory** from the SD Card.



**Directory** holds the directory that will be removed from the SD Card.

**error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**code** is the error or warning code. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**source** describes the origin of the error or warning. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**success** indicates whether or not the operation was successful.

**error out** passes error or warning information out of a VI to be used by other VIs. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

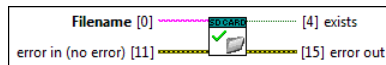
**code** is the error or warning code. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**source** string describes the origin of the error or warning. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

## SD File Exists VI

**Installed With:** Arduino Compatible Compiler for LabVIEW

Returns exists as True if the file held by the Filename input exists on the SD Card.



**Filename** holds the name of the file to be searched for in the SD Card.

**error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**code** is the error or warning code. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**source** describes the origin of the error or warning. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**exists** returns True if the file is found in the SD Card and False otherwise.

**error out** passes error or warning information out of a VI to be used by other VIs. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

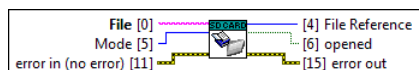
**code** is the error or warning code. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**source** string describes the origin of the error or warning. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

## SD File Open VI

**Installed With:** Arduino Compatible Compiler for LabVIEW

Opens the file held in the File input. The file can be opened to read or read and write, depending on the selection made in the Mode input. Note: The SD library uses 8.3 filenames which have at most 8 characters for files and directories optionally followed by a period and a filename extension of at most three characters. File and directory names are uppercase, although systems that use the 8.3 standard are usually case-insensitive.



**File** holds the file name for the file that will be open. Limited to 8 characters plus a 3 character extension for new files.

**Mode** holds the type of open operation that will be executed.

**error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**Reference** is the reference to the SD Card in use.

**error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**code** is the error or warning code. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**source** describes the origin of the error or warning. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**position** is the current file position returned by the function call.



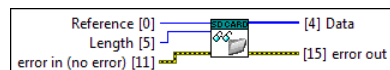


NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

## SD File Read VI

**Installed With:** Arduino Compatible Compiler for LabVIEW

Read the numbers of byte from the file corresponding to the **Length** input.



**Reference** is the reference to the SD Card in use.

**Length** is the length of data to read from the serial port.

**error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**code** is the error or warning code. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**source** describes the origin of the error or warning. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**Data** is a byte array of data read from the serial port. This can be converted to a string if the data being received is ASCII by using the Byte Array to String LabVIEW primitive.

**error out** passes error or warning information out of a VI to be used by other VIs. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**code** is the error or warning code. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

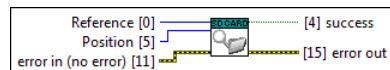
**source** string describes the origin of the error or warning. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

## SD File Seek VI

**Installed With:** Arduino Compatible Compiler for LabVIEW

Seek to a new position in the file, which must be between 0 and the size of the file (inclusive).



**Reference** is the reference to the SD Card in use.

**Position** is the file position sought.

**error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**code** is the error or warning code. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**source** describes the origin of the error or warning. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**success** indicates whether or not the position input was successfully sought.

**error out** passes error or warning information out of a VI to be used by other VIs. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**code** is the error or warning code. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**source** string describes the origin of the error or warning. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

## SD File Size VI

**Installed With:** Arduino Compatible Compiler for LabVIEW

Get the size of the file in number of bytes.





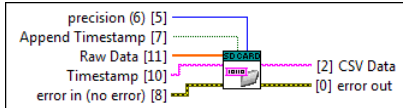


- error out** passes error or warning information out of a VI to be used by other VIs. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.
- status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.
- code** is the error or warning code. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.
- source** string describes the origin of the error or warning. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

## SD Format CSV Data VI

**Installed With:** Arduino Compatible Compiler for LabVIEW

This VI will convert the incoming Data double array input into a CSV line ready to be saved to a SD Card CSV file. It will also append the Timestamp string to the end of the line if the Append Timestamp control is set to true.

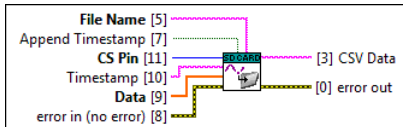


- precision** sets the number of digits of precision the conversion from double to string will perform.
- Append Timestamp** if set to True, the VI will append the Timestamp control data to the end of the CSV line.
- Raw Data** is the array of data that will be converted to the CSV line.
- Timestamp** is the timestamp string, or any other tag, to be appended to the end of the CSV line.
- error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.
- status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.
- code** is the error or warning code. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.
- source** describes the origin of the error or warning. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.
- CSV Data** is the formatted CSV line ready to be saved to a CSV file.
- error out** passes error or warning information out of a VI to be used by other VIs. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.
- status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.
- code** is the error or warning code. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.
- source** string describes the origin of the error or warning. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

## SD Data Logger VI

**Installed With:** Arduino Compatible Compiler for LabVIEW

This VI executes all tasks required by a simple datalogger application. The VI was designed in a manner to be dropped as a subVI inside a loop of on a more elaborate datalogger application. Refer to the Datalogger.vi shipping example. It assumes the utilization of a SD Card shield in conjunction with the Arduino target board.



- File Name** holds the file name for the file that will be open.
- Append Timestamp** Set this control to True if a timestamp is to be appended to the end of the data array line.
- CS Pin** is the SPI Chip Select pin. Refer to the Arduino documentation on SPI at <http://arduino.cc/en/Reference/SDBegin>. Make sure to match the CS Pin that is appropriate for your combination of Arduino target board as well as SD Card shield.
- TimeStamP** is the string that will be appended to the end of the Data Array line and logged to the file. It is only valid if the Append Timestamp control is set to True.
- Data** is a byte array of data read from the serial port. This can be converted to a string if the data being received is ASCII by using the Byte Array to String LabVIEW primitive.
- error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.
- status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.
- code** is the error or warning code. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**source** describes the origin of the error or warning. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**CSV Data** is a copy of the formatted CSV line that was saved to the log file.

**error out** passes error or warning information out of a VI to be used by other VIs. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

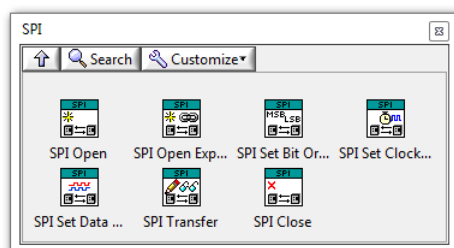
**status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**code** is the error or warning code. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**source** string describes the origin of the error or warning. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

## SPI Palette

**Installed With:** Arduino Compatible Compiler for LabVIEW



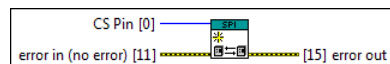
The SPI Palette includes APIs for interfacing to other sensors and devices over an SPI port. This palette only supports use of the SPI peripheral in Master mode.

©2015 TSXperts/Aledyne. All rights reserved

## SPI Open VI

**Installed With:** Arduino Compatible Compiler for LabVIEW

Initializes the SPI bus by setting SCK, MOSI, and SS to outputs, pulling SCK and MOSI low, and SS high. Note that SPI can only be used in Master Mode by this API. For AVR boards, the **CS Pin** is used to configure any digital I/O pin as an output for SPI transactions. For the Arduino Due, the **CS Pin** is configured to be directly managed by the SPI interface and must be one of the allowable CS Pins. Note that once the pin is configured, you can't use it anymore as a general I/O, unless you call SPI Close.vi on the same pin. The only pins that can be configured to be managed by SPI interface are the Arduino Due's Slave Select pins: 4, 10, 52 and 54 (correspond to A0).



**CS Pin** specifies the slave select or chip select pin of the Arduino. This is only relevant for the Arduino Due which supports multiple CS pins.

**error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**code** is the error or warning code. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**source** describes the origin of the error or warning. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**error out** passes error or warning information out of a VI to be used by other VIs. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

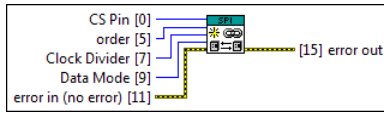
**code** is the error or warning code. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**source** string describes the origin of the error or warning. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

## SPI Open Express VI

**Installed With:** Arduino Compatible Compiler for LabVIEW

Initializes the SPI bus by setting SCK, MOSI, and SS to outputs, pulling SCK and MOSI low, and SS high. Note that SPI can only be used in Master Mode by this API. For AVR boards, the **CS Pin** is used to configure any digital I/O pin as an output for SPI transactions. For the Arduino Due, the **CS Pin** is configured to be directly managed by the SPI interface and must be one of the allowable CS Pins. Note that once the pin is configured, you can't use it anymore as a general I/O, unless you call SPI Close.vi on the same pin. Also configures the clock rate, data mode, and bit order in the same initialization call. The only pins that can be configured to be managed by SPI interface are the Arduino Due's Slave Select pins: 4, 10, 52 and 54 (correspond to A0).



**CS Pin** specifies the slave select or chip select pin of the Arduino. This is only relevant for the Arduino Due which supports multiple CS pins.

**order** specifies the order of the bits shifted out of and into the SPI bus, either LSB First (least-significant bit first) or MSB First (most-significant bit first).

On AVR based boards, **Clock Divider** settings are as follows: 0: Divide By 4 1: Divide by 16 2: Divide by 64 3: Divide by 128 4: Divide by 2 5: Divide by 8 6: Divide by 32 The default setting is Divide by 4, which sets the SPI clock to one-quarter the frequency of the system clock (4 Mhz for the boards at 16 MHz). On the Arduino Due: The system clock can be divided by values from 1 to 255. The default value is 21, which sets the clock to 4 MHz like other Arduino boards.

**Data Mode** is the SPI Data mode specifying the clock polarity and phase.

**error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**code** is the error or warning code. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**source** describes the origin of the error or warning. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**error out** passes error or warning information out of a VI to be used by other VIs. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**code** is the error or warning code. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

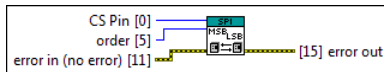
**source** string describes the origin of the error or warning. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

## SPI Set Bit Order VI

**Installed With:** Arduino Compatible Compiler for LabVIEW

Sets the order of the bits shifted out of and into the SPI bus, either LSB First (least-significant bit first) or MSB First (most-significant bit first). For the Arduino Due, the bit order setting is applied only to the device connected to the specified **CS Pin**.



**CS Pin** specifies the slave select or chip select pin of the Arduino. This is only relevant for the Arduino Due which supports multiple CS pins.

**order** specifies the order of the bits shifted out of and into the SPI bus, either LSB First (least-significant bit first) or MSB First (most-significant bit first).

**error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**code** is the error or warning code. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**source** describes the origin of the error or warning. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**error out** passes error or warning information out of a VI to be used by other VIs. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**code** is the error or warning code. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

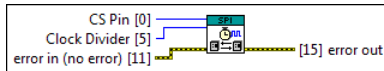
**source** string describes the origin of the error or warning. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

## SPI Set Clock Divider VI

**Installed With:** Arduino Compatible Compiler for LabVIEW

Sets the SPI clock divider relative to the system clock. On AVR based boards, the dividers available are 2, 4, 8, 16, 32, 64 or 128. The default setting is Divide by 4, which sets the SPI clock to one-quarter the frequency of the system clock (4 Mhz for the boards at 16 MHz). On the Arduino Due the system clock can be divided by values from 1 to 255. The default value is 21, which sets the clock to 4 MHz like other Arduino boards. For the Arduino Due, the clock divider setting is applied only to the device connected to the specified **CS Pin**.



**CS Pin** specifies the slave select or chip select pin of the Arduino. This is only relevant for the Arduino Due which supports multiple CS pins.

On AVR based boards, **Clock Divider** settings are as follows: 0: Divide By 4 1: Divide by 16 2: Divide by 64 3: Divide by 128 4: Divide by 2 5: Divide by 2 6: Divide by 32 The default setting is Divide by 4, which sets the SPI clock to one-quarter the frequency of the system clock (4 Mhz for the boards at 16 Mhz). On the Arduino Due: The system clock can be divided by values from 1 to 255. The default value is 21, which sets the clock to 4 Mhz like other Arduino boards.

**error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**code** is the error or warning code. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**source** describes the origin of the error or warning. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**error out** passes error or warning information out of a VI to be used by other VIs. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

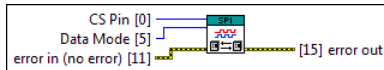
**code** is the error or warning code. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**source** string describes the origin of the error or warning. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

## SPI Set Data Mode VI

**Installed With:** Arduino Compatible Compiler for LabVIEW

Sets the SPI data mode: that is, clock polarity and phase. For the Arduino Due, the clock divider setting is applied only to the device connected to the specified **CS Pin**.



**CS Pin** specifies the slave select or chip select pin of the Arduino. This is only relevant for the Arduino Due which supports multiple CS pins.

**Data Mode** is the SPI Data mode specifying the clock polarity and phase.

**error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**code** is the error or warning code. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**source** describes the origin of the error or warning. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**error out** passes error or warning information out of a VI to be used by other VIs. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

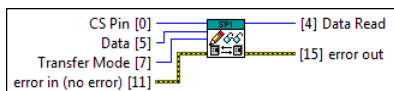
**code** is the error or warning code. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**source** string describes the origin of the error or warning. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

## SPI Transfer VI


**Installed With:** Arduino Compatible Compiler for LabVIEW


Transfers one byte over the SPI bus, both sending and receiving. For all targets, the specified **CS Pin** is activated (pulled low) before the transfer occurs and deactivated (pulled high) when the transfer is finished. There is an automatic 5ms delay after the slave select line is lowered before issuing the SPI transfer. You can use the Transfer Mode parameter to manage the Slave Select line after the transfer. Continue keeps the specified CS pin active (low) after the transfer to allow the send of additional bytes via the SPI Transfer.vi in the same SPI transaction. The last byte to be transferred should be accompanied by the "Last" transfer mode. When a transfer is complete with "Last", the slave select pin returns inactive (high).





**CS Pin** specifies the slave select or chip select pin of the Arduino. This is used on all Arduino boards to specify which CS Pin to enable (lower) during the transfer.




 **Data** is a single byte of data to transmit over the SPI bus.

 **Transfer Mode** specifies whether to leave the CS Pin low after the transfer (Continue) or to raise the CS Pin (Last).


 **error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

 **status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.


 **code** is the error or warning code. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.


 **source** describes the origin of the error or warning. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.


 **Data Read** is the single byte of data read over the SPI bus during the transfer.

 **error out** passes error or warning information out of a VI to be used by other VIs. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.


NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

 **status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

 **code** is the error or warning code. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

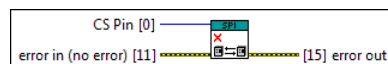
 **source** string describes the origin of the error or warning. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.


NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.


## SPI Close VI

**Installed With:** Arduino Compatible Compiler for LabVIEW


Disables the SPI bus (leaving pin modes unchanged). For the Arduino Due, the specified CS pin is disconnected from the SPI interface and is available again as a general I/O. For AVR boards, the **CS Pin** input has no effect.




 **CS Pin** specifies the slave select or chip select pin of the Arduino. This is only relevant for the Arduino Due which supports multiple CS pins.

 **error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.


NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

 **status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.


NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

 **code** is the error or warning code. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.


NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

 **source** describes the origin of the error or warning. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.


NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

 **error out** passes error or warning information out of a VI to be used by other VIs. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.


NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

 **status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

 **code** is the error or warning code. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

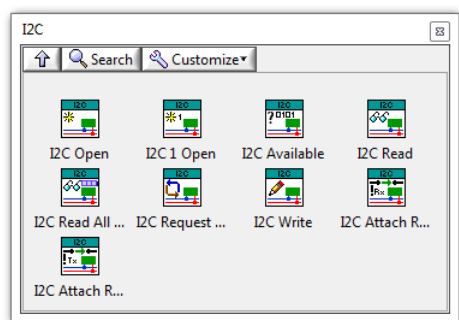
 **source** string describes the origin of the error or warning. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

## I2C Palette

**Installed With:** Arduino Compatible Compiler for LabVIEW





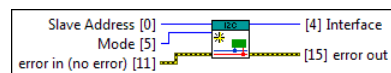
The I2C Palette includes APIs for interfacing to other sensors, devices, and Arduinos over an I2C port. An Arduino I2C port and peripheral using 2 digital pins must be used to interface to this palette. Configuration as a Master or Slave device is supported as well as I2C receive interrupts.

©2015 TSXperts/Aledyne. All rights reserved

## I2C Open VI

**Installed With:** Arduino Compatible Compiler for LabVIEW

Initiate the I2C interface (SDA/SCL) and joins the I2C bus as a master or slave. If **Mode** is set to Master, the **Slave Address** is ignored. If **Mode** is set to Slave, the Slave Address will be applied.



**Slave Address** defines the slave address to apply when in Slave mode.

**Mode** defines whether to make this node a master or a slave. If a Slave is specified, the **Slave Address** should be set appropriately.

**error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**code** is the error or warning code. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**source** describes the origin of the error or warning. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**Interface** is the reference to the I2C port. This should be passed into subsequent I2C port VIs.

**error out** passes error or warning information out of a VI to be used by other VIs. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

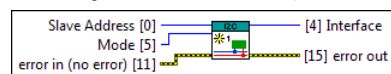
**code** is the error or warning code. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**source** string describes the origin of the error or warning. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

## I2C 1 Open VI

**Installed With:** Arduino Compatible Compiler for LabVIEW

Initiate the I2C 1 interface (Due only - SDA1/SCL1) and joins the I2C bus as a master or slave. The Arduino Due has 2 I2C interfaces 20 (SDA), 21 (SCL), SDA1, SCL1. If **Mode** is set to Master, the **Slave Address** is ignored. If **Mode** is set to Slave, the Slave Address will be applied.



**Slave Address** defines the slave address to apply when in Slave mode.

**Mode** defines whether to make this node a master or a slave. If a Slave is specified, the **Slave Address** should be set appropriately.

**error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**code** is the error or warning code. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**source** describes the origin of the error or warning. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more

 **error out** passes error or warning information out of a VI to be used by other VIs. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

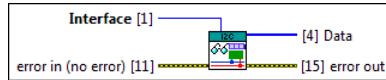
**code** is the error or warning code. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**source** string describes the origin of the error or warning. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

## I2C Read All Bytes VI

**Installed With:** Arduino Compatible Compiler for LabVIEW

Reads all available bytes from the specified I2C interface.



**Interface** is the reference to the I2C port.

**error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**code** is the error or warning code. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**source** describes the origin of the error or warning. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**Data** is a byte array of data read from the I2C port. This can be converted to a string if the data being received is ASCII by using the Byte Array to String LabVIEW primitive.

**error out** passes error or warning information out of a VI to be used by other VIs. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

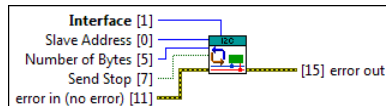
**code** is the error or warning code. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**source** string describes the origin of the error or warning. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

## I2C Request From VI

**Installed With:** Arduino Compatible Compiler for LabVIEW

Used by the master to request bytes from a slave device. The bytes may then be retrieved with the I2C Read.vi and I2C Read All Bytes.vi functions.



**Slave Address** is a the address of the slave device to request data from.

**Interface** is the reference to the I2C port.

**Number of Bytes** to request from the slave device specified.

**Send Stop** set to true will send a stop message after the request releasing the I2C bus. If false, a restart message is sent after the request. The bus will not be released, which prevents another master device from requesting between messages. This allows one master device to send multiple requests while in control.

**error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**code** is the error or warning code. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**source** describes the origin of the error or warning. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**error out** passes error or warning information out of a VI to be used by other VIs. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

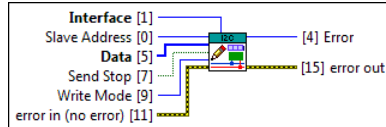
**status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

- 132** **code** is the error or warning code. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.
- 13c** **source** string describes the origin of the error or warning. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

## I2C Write Array VI

**Installed With:** Arduino Compatible Compiler for LabVIEW

Writes data from a slave device in response to a request from a master, or queues bytes for transmission from a master to slave device.

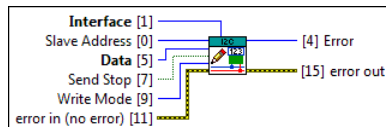


- 08** **Slave Address** is a the address of the slave device to send data to.
- 08** **Interface** is the reference to the I2C port.
- 08** **Data** to send to the slave device.
- TF** **Send Stop** set to true will send a stop message after the transfer releasing the I2C bus. If false, a restart message is sent after the transfer. The bus will not be released, which prevents another master device from sending between messages. This allows one master device to send multiple transmissions while in control.
- 0** **Write Mode** defines how to handle the start and end transmission calls.  
Begin and End: Calls Begin Transmission before queuing the input data, then calls End Transmission after queuing the data to force all data to be sent and forces a stop or restart.  
Begin and Queue: Begin Transmission and queues the input data only. Does not send a stop or restart so that additional write calls can be made before ending the transmission.  
Queue and End: Queues the input data then calls End Transmission after queuing the data to force all data to be sent and forces a stop or restart. Assumes a Begin Transmission has been previously called.  
Queue Only: Only Queues the input data and does NOT call Begin or End Transmission before or after queuing the data.
- 132** **error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.
- TF** **status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.
- 132** **code** is the error or warning code. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.
- 13c** **source** describes the origin of the error or warning. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.
- 132** **Error** indicates the status of the transmission. 0: success 1: data too long to fit in transmit buffer 2: received NACK on transmit of address 3: received NACK on transmit of data 4: other error
- 132** **error out** passes error or warning information out of a VI to be used by other VIs. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.
- TF** **status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.
- 132** **code** is the error or warning code. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.
- 13c** **source** string describes the origin of the error or warning. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

## I2C Write Byte VI

**Installed With:** Arduino Compatible Compiler for LabVIEW

Writes data from a slave device in response to a request from a master, or queues bytes for transmission from a master to slave device.



- 08** **Slave Address** is a the address of the slave device to send data to.
- 08** **Interface** is the reference to the I2C port.
- 08** **Data** to send to the slave device.
- TF** **Send Stop** set to true will send a stop message after the transfer releasing the I2C bus. If false, a restart message is sent after the transfer. The bus will not be released, which prevents another master device from sending between messages. This allows one master device to send multiple transmissions while in control.
- 0** **Write Mode** defines how to handle the start and end transmission calls.  
Begin and End: Calls Begin Transmission before queuing the input data, then calls End Transmission after queuing the data to force all data to be sent and forces a stop or restart.  
Begin and Queue: Begin Transmission and queues the input data only. Does not send a stop or restart so that additional write calls can be made before ending the transmission.  
Queue and End: Queues the input data then calls End Transmission after queuing the data to force all data to be sent and forces a stop or restart. Assumes a Begin Transmission has been previously called.  
Queue Only: Only Queues the input data and does NOT call Begin or End Transmission before or after queuing the data.
- 132** **error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.
- TF** **status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**U32** **code** is the error or warning code. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**abc** **source** describes the origin of the error or warning. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**U32** **Error** indicates the status of the transmission. 0: success 1: data too long to fit in transmit buffer 2: received NACK on transmit of address 3: received NACK on transmit of data 4: other error

**Ex** **error out** passes error or warning information out of a VI to be used by other VIs. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**TF** **status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

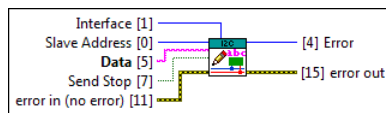
**U32** **code** is the error or warning code. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**abc** **source** string describes the origin of the error or warning. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

## I2C Write String VI

**Installed With:** Arduino Compatible Compiler for LabVIEW

Writes data from a slave device in response to a request from a master, or queues bytes for transmission from a master to slave device. Automatically calls Begin Transmission before queuing the data and End Transmission after.



**U8** **Slave Address** is a the address of the slave device to send data to.

**U8** **Interface** is the reference to the I2C port.

**abc** **Data** to send to the slave device.

**TF** **Send Stop** set to true will send a stop message after the transfer releasing the I2C bus. If false, a restart message is sent after the transfer. The bus will not be released, which prevents another master device from sending between messages. This allows one master device to send multiple transmissions while in control.

**Ex** **error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**TF** **status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**U32** **code** is the error or warning code. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**abc** **source** describes the origin of the error or warning. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**U32** **Error** indicates the status of the transmission. 0: success 1: data too long to fit in transmit buffer 2: received NACK on transmit of address 3: received NACK on transmit of data 4: other error

**Ex** **error out** passes error or warning information out of a VI to be used by other VIs. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**TF** **status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**U32** **code** is the error or warning code. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

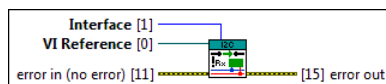
**abc** **source** string describes the origin of the error or warning. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

## I2C Attach Receive Interrupt VI

**Installed With:** Arduino Compatible Compiler for LabVIEW

Specifies a callback VI to call in **VI Reference** when a slave device receives a transmission from a master. The callback VI will be called when the slave receives data and should take a single U16 parameter (the number of bytes read from the master) and return nothing. Inside the attached VI, and Delay calls won't work and the value returned by Tick Count vis will not increment. Serial data received while in the function may be lost. To transfer data in and out of the callback VI, global variables must be used.



**D** **VI Reference** is the reference to the callback VI that will run when the interrupt occurs.

**U8** **Interface** is the reference to the I2C port.

**Ex** **error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**TF** **status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.  
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to

help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**I324** **code** is the error or warning code. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**I344** **source** describes the origin of the error or warning. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**I345** **error out** passes error or warning information out of a VI to be used by other VIs. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**I346** **status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**I322** **code** is the error or warning code. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

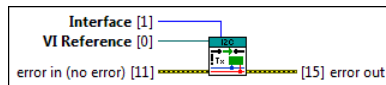
**I343** **source** string describes the origin of the error or warning. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

## I2C Attach Request Interrupt VI

**Installed With:** Arduino Compatible Compiler for LabVIEW

Specifies a callback VI to call in **VI Reference** when a master requests data from this slave device. The callback VI will be called when the master requests data and should not take in any parameters and returns nothing. Inside the attached VI, and Delay calls won't work and the value returned by Tick Count vis will not increment. Serial data received while in the function may be lost. To transfer data in and out of the callback VI, global variables must be used.



**I347** **VI Reference** is the reference to the callback VI that will run when the interrupt occurs.

**I348** **Interface** is the reference to the I2C port.

**I349** **error in** can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**I346** **status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**I322** **code** is the error or warning code. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**I344** **source** describes the origin of the error or warning. Right-click the **error in** control on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**I345** **error out** passes error or warning information out of a VI to be used by other VIs. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**I346** **status** is TRUE (X) if an error occurred or FALSE (checkmark) to indicate a warning or that no error occurred. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**I322** **code** is the error or warning code. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

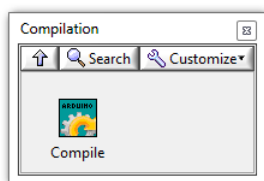
NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

**I343** **source** string describes the origin of the error or warning. Right-click the **error out** indicator on the front panel and select **Explain Error** or **Explain Warning** from the shortcut menu for more information about the error.

NOTE: error wires are only used for flow control when deployed onto an Arduino target and the data within the cluster cannot be written or read. The error wires should only be used to help control dataflow in a program since cluster datatypes are not supported due to low memory constraints.

## Compilation Palette

**Installed With:** Arduino Compatible Compiler for LabVIEW



This palette contains an API in order to programmatically call the Arduino Compatible Compiler for LabVIEW to compile and download VIs from a customized VI or user interface. There is also a VI that can be called from a [command line interface](#) in order to instantiate the compiler from another program or programming language.

©2015 TSXperts/Aledyne. All rights reserved

## Compile VI


**Installed With:** Arduino Compatible Compiler for LabVIEW


Calls the Arduino Compatible Compiler for LabVIEW and either compiles only or compiles and downloads the specified VI to the specified Arduino target board. If the specified board has multiple CPU types,


then it must be identified in **CPU**. Some Boards only have one CPU types, in which the CPU field can be left blank.


 Compile

 datatype\_images\cenum.gif **Compile Options** specifies either compile only or compile and download to an Arduino target.


 datatype\_images\cpath.gif **VI Path** specifies the VI to compile (or download) to the specified target.

 datatype\_images\cstr.gif **COM Port** specifies the Serial port of the Arduino target.


 datatype\_images\cstr.gif **Architecture** specifies the architecture of the arduino core, either **avr** or **sam**.


 datatype\_images\cstr.gif **Board** specifies the type of arduino board and must be one of the following:


- yun
- uno
- diecimila
- nano
- mega
- megaADK
- leonardo
- micro
- esplora
- mini
- ethernet
- fio
- bt
- LilyPadUSB
- lilypad
- pro
- atmegang
- robotControl
- robotMotor
- arduino\_due\_x\_dbg
- arduino\_due\_x


 datatype\_images\cstr.gif **CPU** specifies the CPU variation of the specified board if more than one CPU is available. This is only relevant to certain boards and should be left blank unless the board has more than one CPU type. Refer to the Arduino Compatible Compile for LabVIEW "Board" menu for a list of boards that have more than one CPU type. Possible variations include:


- atmega328
- atmega168
- atmega8
- atmega2560
- atmega1280
- 16MHzatmega328
- 8MHzatmega328
- 16MHzatmega168
- 8MHzatmega168


 datatype\_images\cstr.gif **Package** is the identifier of the vendor (the first level folders inside the hardware directory). Default arduino boards use arduino.


 datatype\_images\ccclst.gif The **error in** cluster can accept error information wired from VIs previously called. Use this information to decide if any functionality should be bypassed in the event of errors from other VIs. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.


 datatype\_images\cbool.gif The **status** boolean is either TRUE (X) for an error, or FALSE (checkmark) for no error or a warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.


 datatype\_images\ci32.gif The **code** input identifies the error or warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.


 datatype\_images\cstr.gif The **source** string describes the origin of the error or warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

 datatype\_images\istr.gif **status** provides status information after a compile and/or download is complete. This is usually the output physical space that the VI was compiled down to. If there is an error during compilation, details will be provided on the **error out** terminal.

 datatype\_images\icclst.gif The **error out** cluster passes error or warning information out of a VI to be used by other VIs. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

 datatype\_images\ibool.gif The **status** boolean is either TRUE (X) for an error, or FALSE (checkmark) for no error or a warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

 datatype\_images\ii32.gif The **code** input identifies the error or warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

 datatype\_images\istr.gif The **source** string describes the origin of the error or warning. The pop-up option **Explain Error** (or Explain Warning) gives more information about the error displayed.

## Command Line Interface

**Installed With:** Arduino Compatible Compiler for LabVIEW



```

cmd.exe - Shortcut
Microsoft Windows [Version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Windows\SysWOW64>cd C:\Program Files (x86)\National Instruments\LabVIEW 2014

C:\Program Files (x86)\National Instruments\LabVIEW 2014>labview.exe "C:\Program
Files (x86)\National Instruments\LabVIEW 2014\vi.lib\Aledyne-TSXperts\Arduino C
ompatible Compiler for LabVIEW\alvcompiler.vi" -- verify -arch avr -board uno -p
ath "C:\Program Files (x86)\National Instruments\LabVIEW 2014\examples\Aledyne-I
SXperts\Arduino Compatible Compiler for LabVIEW\Analog\Analog Input - Temperatur
e.vi"

C:\Program Files (x86)\National Instruments\LabVIEW 2014>
Sketch uses 1,724 bytes (5%) of program storage space. Maximum is 32,256 bytes.
Global variables use 17 bytes (0%) of dynamic memory, leaving 2,031 bytes for lo
cal variables. Maximum is 2,048 bytes.

```

The Arduino Compatible Compiler for LabVIEW can be run from a command prompt by calling "alvcompiler.vi" located at [LabVIEW]\vi.lib\Aledyne-TSXperts\Arduino Compatible Compiler for LabVIEW\alvcompiler.vi"

From any directory:

```
> "C:\Program Files (x86)\National Instruments\LabVIEW 2014\labview.exe" "... \Arduino Compatible Compiler for
LabVIEW\alvcompiler.vi" -- [compile|download] [-arch architecture] [-board BOARDTYPE] [-cpu CPUTYPE] [-port
PORTNAME] [-path FILE.vi]
```

or from the LabVIEW directory:

```
C:\Program Files (x86)\National Instruments\LabVIEW 2014> labview.exe "... \Arduino Compatible Compiler for
LabVIEW\alvcompiler.vi" -- [compile|download] [-arch ARCHITECTURE] [-board BOARDTYPE] [-cpu CPUTYPE] [-port
PORTNAME] [-path FILE.vi]
```

- **compile** or **download** specifies whether to compile the VI only or compile then download it.
- **-package** specifies the vendor (the first level folders inside the hardware directory). Default arduino boards use **arduino**. If no package is specified, arduino is used.
- **-arch** specifies the architecture. Use either **avr** or **sam**. The format would be **-arch avr** for example.
- **-board** is the actual board to use. For example, **uno** for the Arduino Uno, **diecimila** for the Arduino Duemilanove or Diecimila, **mega** for the Arduino Mega. The format would be **-board uno** for example.  
Other variants include: yun, uno, diecimila, nano, mega, megaADK, leonardo, micro, esplora, mini, ethernet, fio, bt, LilyPadUSB, lilypad, pro, atmegang, robotControl, robotMotor, arduino\_due\_x\_dbg, and arduino\_due\_x
- **-cpu** specifies a cpu variant of the specified board. This is only relevant to certain boards and should NOT be included unless the board has more than one CPU type. Refer to the Arduino Compatible Compile for LabVIEW "Board" menu for a list of boards that have more than one CPU type. For example, for the nano the following would be included **-cpu atmega168** to select the mega168 variant of the Arduino Nano board.  
Other variants include: atmega328, atmega168, atmega8, atmega2560, atmega1280, 16MHzatmega328, 8MHzatmega328, 16MHzatmega168, 8MHzatmega168
- **-port** specifies the serial port to use. This is only requires if the download option is selected. The format would be **-port COM1** for example.
- **-path** specifies the VI path to compile or download. The format would be **-path [FILEPATH].vi** for example.

NOTE: The order of commands in the command string does not matter.

A complete examples is as follows:

```
labview.exe "C:\Program Files (x86)\National Instruments\LabVIEW 2014\vi.lib\Aledyne-TSXperts\Arduino Compatible Compiler for
LabVIEW\alvcompiler.vi" -- verify -arch avr -board uno -path "C:\Program Files (x86)\National Instruments\LabVIEW 2014
\examples\Aledyne-TSXperts\Arduino Compatible Compiler for LabVIEW\Analog\Analog Input - Temperature.vi"
```

©2015 TSXperts/Aledyne. All rights reserved

## Examples

**Installed With:** Arduino Compatible Compiler for LabVIEW



The Arduino Compatible Compiler for LabVIEW ships with various different examples to get you up and running quickly. These examples have been tested on numerous Arduino platforms, including the Uno, Mega, Leonardo, Yun, and Due. If an example has a front panel control, its data can be changed and the VI can be re-compiled and downloaded to reflect the latest changes without needing to save the

VI first. The following examples have been included for your convenience:

### 1. Digital

- a. [Polling a button Example](#): This example shows how to read a digital input on an Arduino target. This example assumes a switch is wired to the digital input, therefore it implements debouncing when the input changes to a low state. A digital output is toggled when the switch is pressed. Instead of wiring an LED, the onboard LED wired to digital pin 13 can also be used.
- b. [Blink LED Example](#): This example shows how to set an Arduino digital pin to an output, then toggle the voltage between low (0V) and high (5V) to drive an LED on and off. Instead of wiring an LED, the onboard LED wired to digital pin 13 can also be used.

### 2. Analog

- a. [Temperature Example](#): This example shows how to read the voltage of an Arduino analog input pin which has a LM-35 or TMP35 temperature sensor connected to it, and converts the acquired voltage to Celsius. If the temperature is larger than the set threshold on the front panel, the digital output specified is set to high. Otherwise, it is set to low. This can be connected to an LED to see that the analog input is being acquired properly. In order to use this example, a temperature sensor must be connected to an analog input specified on the front panel. Instead of wiring an LED, the onboard LED wired to digital pin 13 can also be used.
- b. [3 Axis Accelerometer Example](#): This example shows how to acquire analog data from a 3-axis accelerometer that is powered from 3.3V (such as an ADXL335) and convert to roll and pitch to determine when the orientation is within a given range. This is similar as to how mobile devices detect screen orientation for portrait and landscape modes.
- c. [PWM Example](#): Demonstrates how to use a PWM output on a digital pin to vary the brightness of an LED. This can also be used to vary the speed of a motor, for example. This can be run on any Arduino board that supports a PWM on a digital output pin. The Due supports true DAC outputs on pins 66 and 67 and this same example can be used to control the analog outputs of those DACs.

### 3. Interrupts

- a. [Interrupt on Digital Edge Example](#): This example shows how to setup a change on a digital input to trigger an interrupt to call into a callback VI. In the callback VI outputs I/O can be modified directly, or global variables can be used to update variables shared with the main VI. This example demonstrates sharing data between the interrupt (callback VI) and the main VI using global variables. A counter is updated in the interrupt, and every 5 edges, the output state of a digital output pin is changed. In the main VI, the same output pin is reset between edge transitions.
- b. [Timer Interrupt Example](#): This example shows how to setup a microsecond based timer to trigger an interrupt to call into a callback VI. In the callback VI outputs I/O can be modified directly, or global variables can be used to update variables shared with the main VI. This example uses the onboard LED on pin 13.

### 4. Tone Generation

- a. [Play Song Example](#): This example shows how to use the tone generator on a digital pin to play a song through a piezo buzzer. The tempo of the song can also be adjusted. The use of a subVI is also demonstrated as a lookup table to obtain the frequency of a specified tone. The tempo of the song can be changed by adjusting the control on the front panel (lower the number to make the song play faster) and re-download to observe the changes to the song.

### 5. Serial

- a. [Monitoring GUI Example](#): This example shows best practices for a remote monitoring system with two VIs, one running on a host PC and the other running embedded in the Arduino compatible target. The example includes two-way serial communication between the host PC VI and the Arduino compatible target VI. The host PC monitors and charts a specific variable of interest that is updated in the embedded Arduino target VI. The host PC VI also includes the ability to pause and resume the sending of data points by the Arduino compatible target VI. The target VI generates a point by point sine wave that is charted by the host PC VI.

### 6. I2C LCD

- a. [4 Line LCD Example](#): This example demonstrates how to initialize and write data to an I2C based LCD screen. To use the I2C LCD library, most parallel LCDs can be used with a low cost Sainsmart LCD I2C Adapter. The purpose of the adapter is to cut down the amount of conductors needed by using a serial I2C interface to the LCD. This example assumes a Sainsmart I2C LCD or I2C Adapter is being used. This example has been tested with the following LCDs: SainSmart IIC/I2C/TWI Serial 2004 20x4 LCD Module Shield. The LiquidCrystal\_I2C Express.vi initialization is demonstrated here, which hardcodes the interface pins between the LCD module and the parallel LCD.
- b. [2 Line LCD Example](#): This example demonstrates how to initialize and write data to an I2C based LCD screen, specifically a 2 Line, 16 Character LCD. To use the I2C LCD library, most parallel LCDs can be used with a low cost Sainsmart LCD I2C Adapter. The purpose of the adapter is to cut down the amount of conductors needed by using a serial I2C interface to the LCD. This example assumes a Sainsmart I2C LCD or I2C Adapter is being used. This example has been tested with the following LCDs: SainSmart IIC/I2C/TWI Serial 2004 16x2 LCD Module Shield.
- c. [Simultaneous LCD Example](#): This example demonstrates how to initialize and write data to multiple I2C LCD screens at the same time. Separate LCD instances are used to address each LCD in the same execution loop. Each LCD must have a different I2C address since they are wired to the same I2C bus. The 4x20 LCD uses I2C address 0x3F and the 2x16 LCD uses I2C Address 0x27. This example has been tested with the following LCDs: SainSmart IIC/I2C/TWI Serial 2004 20x4 LCD Module Shield and SainSmart IIC/I2C/TWI Serial 2004 16x2 LCD Module Shield.

### 7. RGB LED

- a. [Chaser Example](#): This example demonstrates a theater light chaser on a 1-wire RGB LED strip using a WS2811 or WS2812 controller. The color of all LEDs can be preset as well as the number of lights to cycle in the light chase. The time delay can also be adjusted to change the dramatic effect.
- b. [Rainbow Example](#): This example demonstrates a rainbow effect on a 1-wire RGB LED strip using a WS2811 or WS2812 controller. Each LED in the strip is gradually changed from red to green, then to blue with a blend of colors in between. The LED color changing effect is staggered down the strip so not all LEDs are changed at the same time. The time delay can also be adjusted to change how fast the color is changed.

### 8. SD Card

- a. [Read Data Example](#): This example demonstrates how to read data previously written to an SD Card and to transfer this data via serial communication. The project includes two VIs, one running on a host PC and the other running embedded in the Arduino compatible target.
- b. [Log Data Example](#): This example demonstrates how to initialize and write data to an SD Card Shield. Once this example is executed successfully, you can run the SD Card - Read Data.vi shipping example to retrieve the data saved in the SD Card by this

example and plot it on the screen of a host VI.

#### 9. SPI

- a. [MAX6675 Thermocouple to Digital Converter Example](#): This example demonstrates how to read the temperature from a MAX6675 SPI K-Type Thermocouple to Digital converter. The temperature is read in Celsius, then converted to Fahrenheit and written to the serial port every 200ms. On AVR targets only one Chip Select pin (pin 10) is supported by the SPI peripheral, however on Due targets, other CS Pins can be selected on the front panel before downloading.

#### 10. I2C

- a. [DS1307 Real Time Clock Example](#): This example demonstrates how to set and read the current time from a DS1307 Real-Time Clock (RTC). This RTC is used on the Arduino Datalogger Shield, which also comes equipped with an SD card slot. To set the current time, enable the "Set" control and populate the date structure with the current date/time, then download the code first. Then turn off "Set" and download again to read the time only. The current date/time will be maintained after power cycle if the battery is installed in the shield.
- b. [Master/Slave Example](#): This example demonstrates how to send data between 2 Arduino boards using the I2C interface. The Master (transmitting) end of the communication writes its loop and timing information to the Slave as a String. The Slave (receiving) end of the communication registers a callback VI to be run whenever an I2C receive interrupt occurs making this an interrupt driven message handler. When an I2C message comes in, the data is formatted and written to the serial terminal for demo purposes only.

#### 11. EEPROM

- a. [EEPROM Read Write Example](#): This example demonstrates how to read and write data to the Arduino EEPROM. Refer to the documentation of your specific Arduino target for more information on its EEPROM. This example retrieves the size of your Arduino EEPROM, writes values to two independent EEPROM addresses and read those two values back. The three data points are displayed on an I2C based LCD screen. Note: the Arduino Due does not have an EEPROM so this example will no work on that target.

#### 12. Memory Optimization

- a. [Memory Optimization SubVI Example](#): This example demonstrates how to pass large data sets to subVIs using Global Variables. This is the preferred method for passing large data sets to subVIs as it avoids the creation of extra memory copies of those data sets inside the subVIs. Refer to the Important Considerations Before Start Creating Arduino VIs section of the Quick Start Guide for more information on Memory Optimization techniques.
- b. [Saving Read-Only Data to Program Memory Example](#): This example demonstrates how to store data in flash (program) memory instead of SRAM and read it back within a running program. This provides a way to tell the compiler to put a large array into flash memory instead of into SRAM, where it would normally go. The data stored in program memory is only available as read-only to your program. This is a good way to get use of more memory for things like lookup tables. You must use the Read Program Memory functions to retrieve the data from your program at run-time.

#### 13. Debugging

- a. [Memory Crash Example](#): This example demonstrates how to use the Debug Tool API VI in conjunction with the Arduino Compatible Compiler for LabVIEW Serial Monitor to debug an embedded Arduino VI. It also demonstrate how to look for memory leaks on an application that doesn't generate the expected results even though it doesn't generate a compilation error.

#### 14. Addons

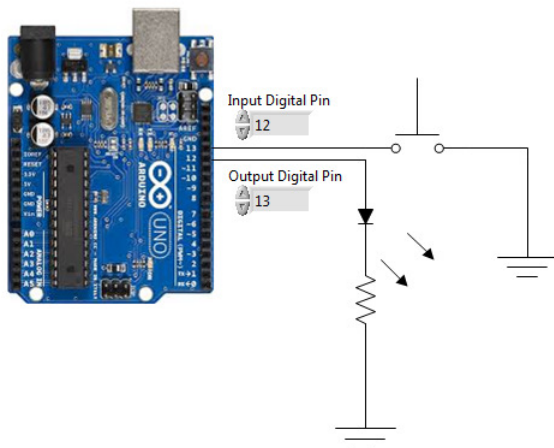
- a. [Digilent Analog Shield Example](#): This example demonstrates how to utilize the Digilent Analog Shield to acquire analog data from a signal source connected to an input analog pin on the shield and output the acquired data to an analog output pin on the same shield. The analog output data will be exactly the same as the acquired analog input data. This example utilizes AI 0 and AO 0 on the shield, but any of the four available input/output channels can be used.

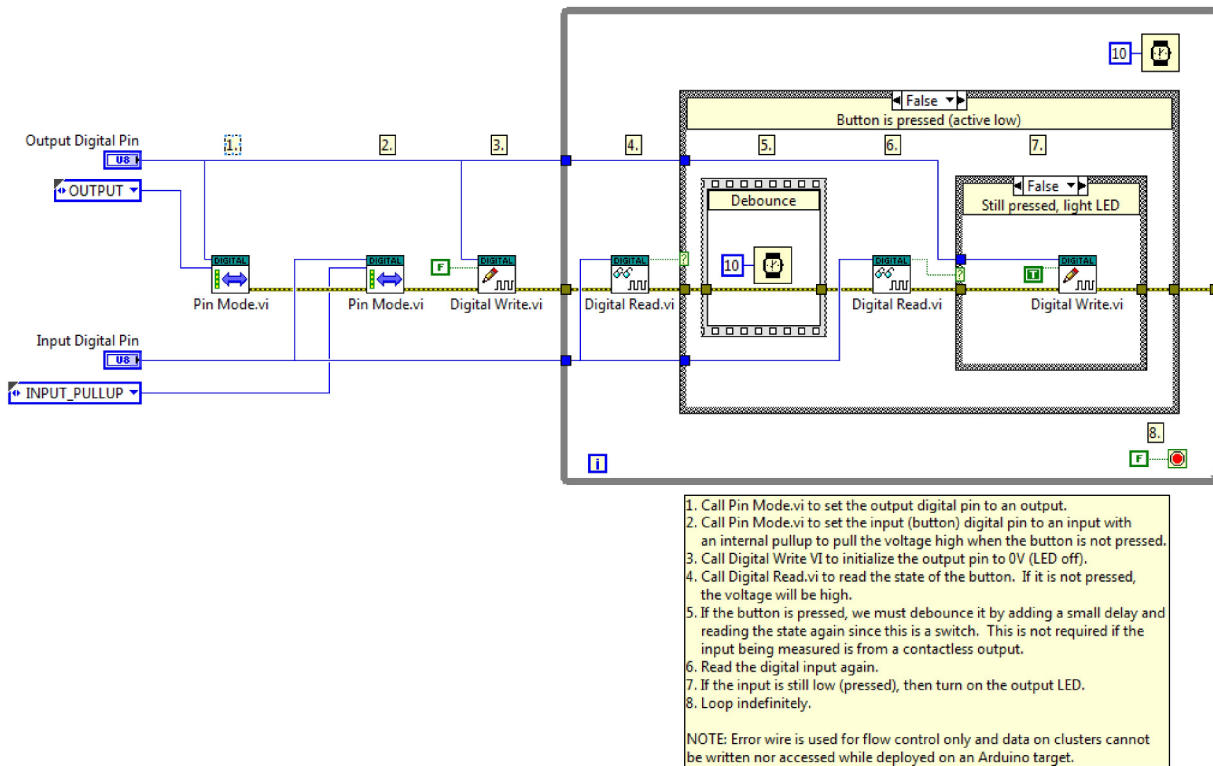
©2015 TSXperts/Aledyne. All rights reserved

### Digital Input - Polling a Button Example

**Installed With:** Arduino Compatible Compiler for LabVIEW

This example shows how to read a digital input on an Arduino target. This example assumes a switch is wired to the digital input, therefore it implements debouncing when the input changes to a low state. A digital output is toggled when the switch is pressed. Instead of wiring an LED, the onboard LED wired to digital pin 13 can also be used.



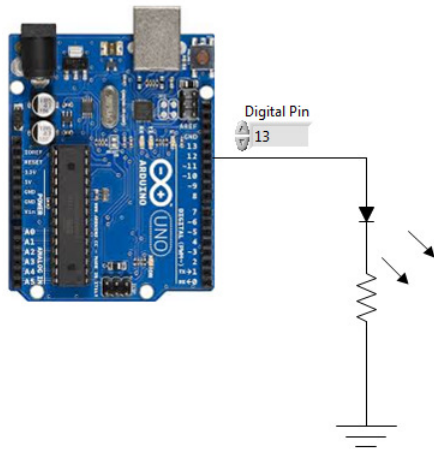


©2015 TSXperts/Aledyne. All rights reserved

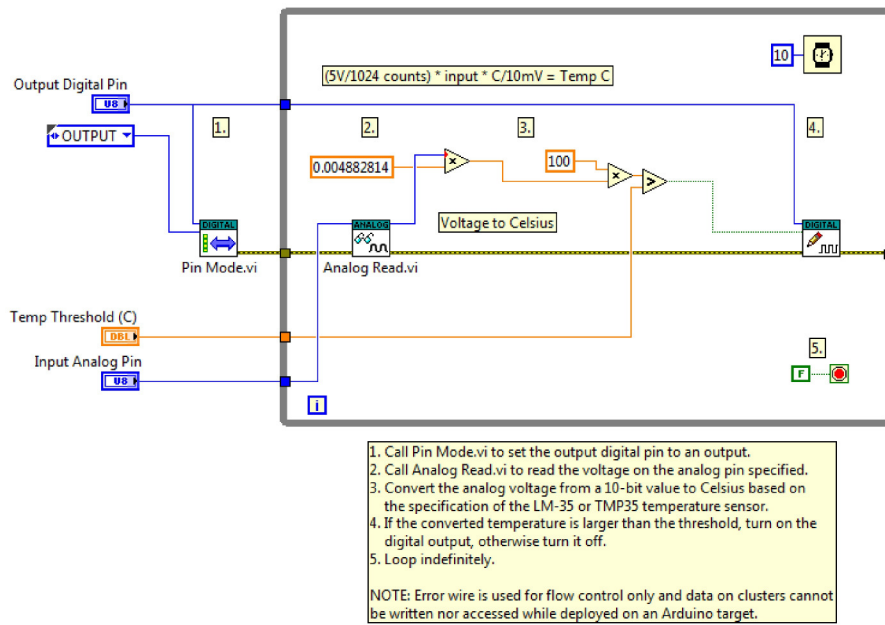
### Digital Output - Blink LED Example

Installed With: Arduino Compatible Compiler for LabVIEW

This example shows how to set an Arduino digital pin to an output, then toggle the voltage between low (0V) and high (5V) to drive an LED on and off. Instead of wiring an LED, the onboard LED wired to digital pin 13 can also be used.





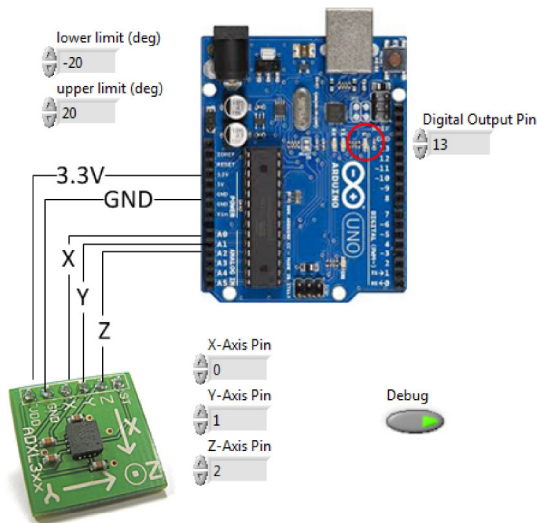


©2015 TSXperts/Aledyne. All rights reserved

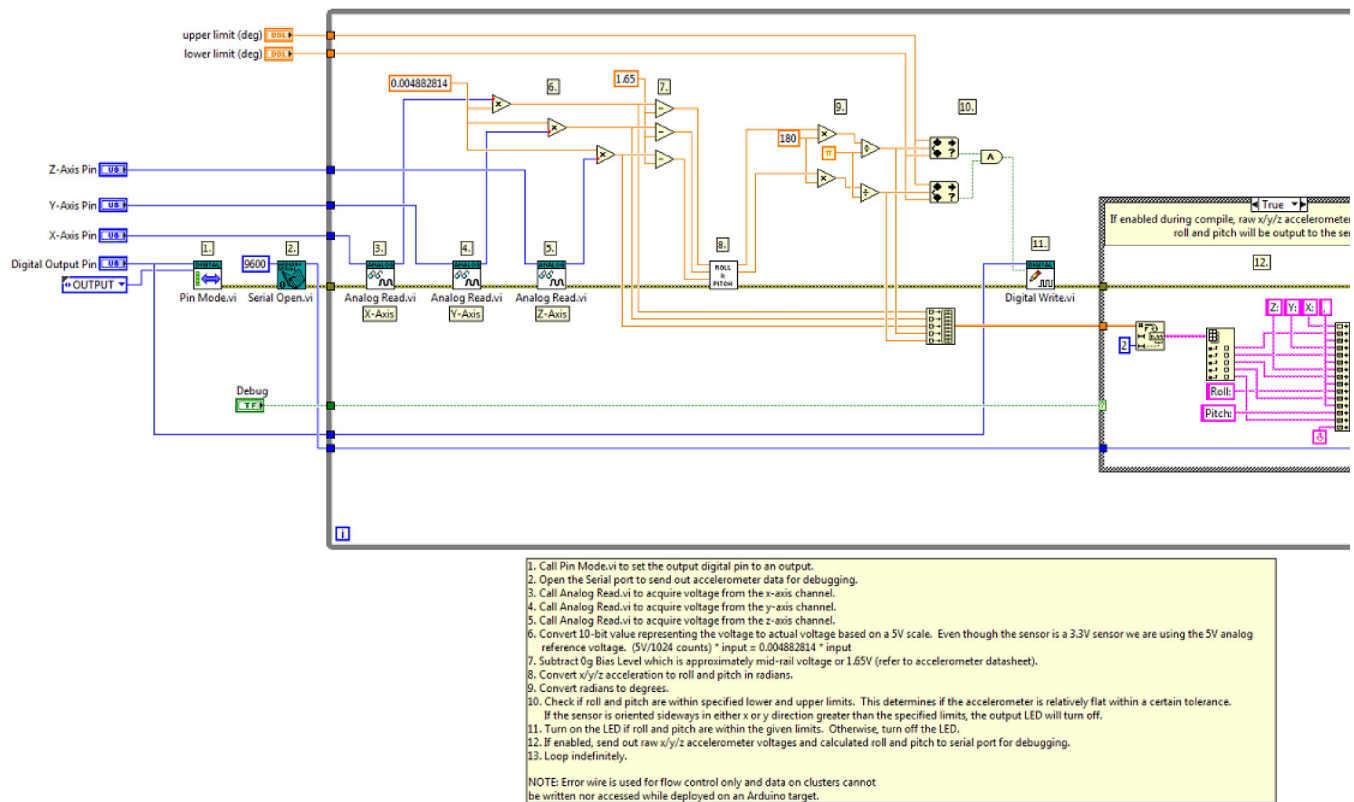
### Analog Input - 3 Axis Accelerometer Example

Installed With: Arduino Compatible Compiler for LabVIEW

This example shows how to acquire analog data from a 3-axis accelerometer that is powered from 3.3V (such as an ADXL335) and convert to roll and pitch to determine when the orientation is within a given range. This is similar as to how mobile devices detect screen orientation for portrait and landscape modes.





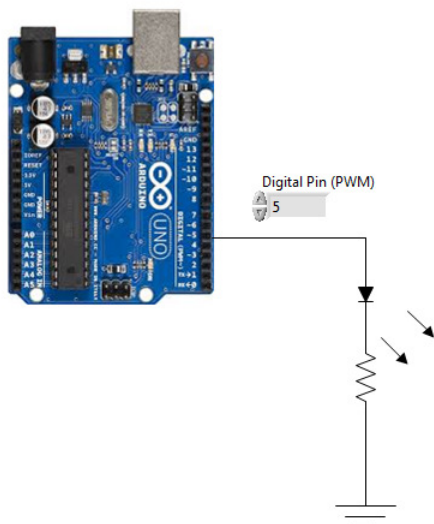


©2015 TSXperts/Aledyne. All rights reserved

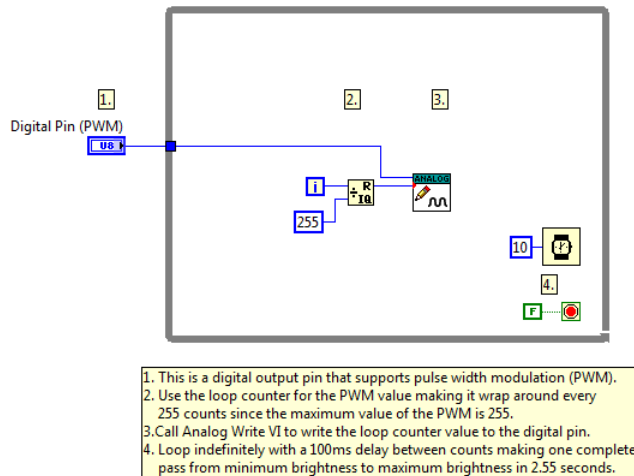
### Analog Output - PWM Example

Installed With: Arduino Compatible Compiler for LabVIEW

Demonstrates how to use a PWM output on a digital pin to vary the brightness of an LED. This can also be used to vary the speed of a motor, for example. This can be run on any Arduino board that supports a PWM on a digital output pin. The Due supports true DAC outputs on pins 66 and 67 and this same example can be used to control the analog outputs of those DACs.





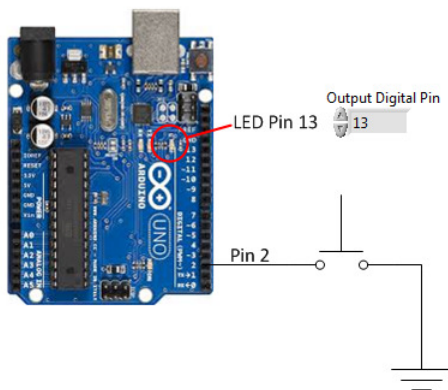


©2015 TSXperts/Aledyne. All rights reserved

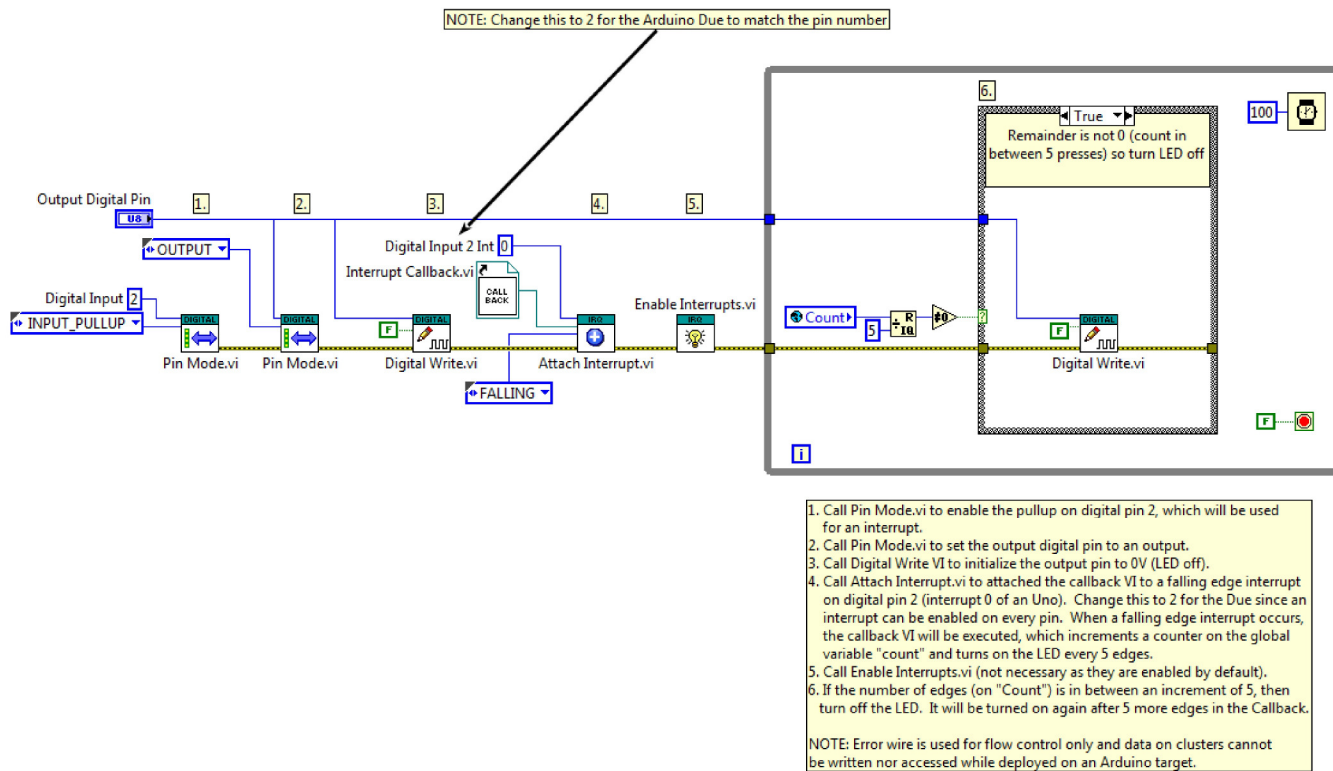
### Interrupt - On Digital Input Edge Example

**Installed With:** Arduino Compatible Compiler for LabVIEW

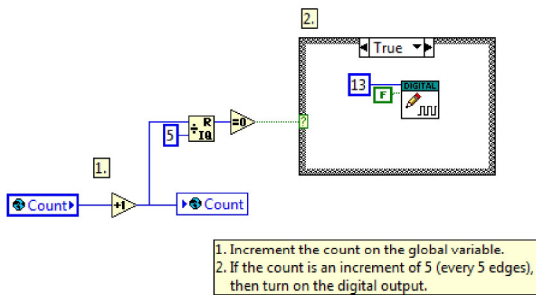
This example shows how to setup a change on a digital input to trigger an interrupt to call into a callback VI. In the callback VI outputs I/O can be modified directly, or global variables can be used to update variables shared with the main VI. This example demonstrates sharing data between the interrupt (callback VI) and the main VI using global variables. A counter is updated in the interrupt, and every 5 edges, the output state of a digital output pin is changed. In the main VI, the same output pin is reset between edge transitions.



### Main VI:



## Callback VI:

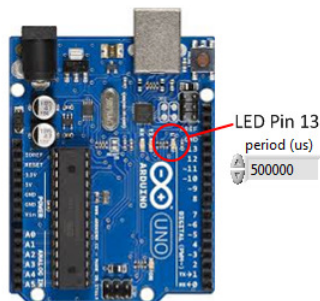


©2015 TSXperts/Aledyne. All rights reserved

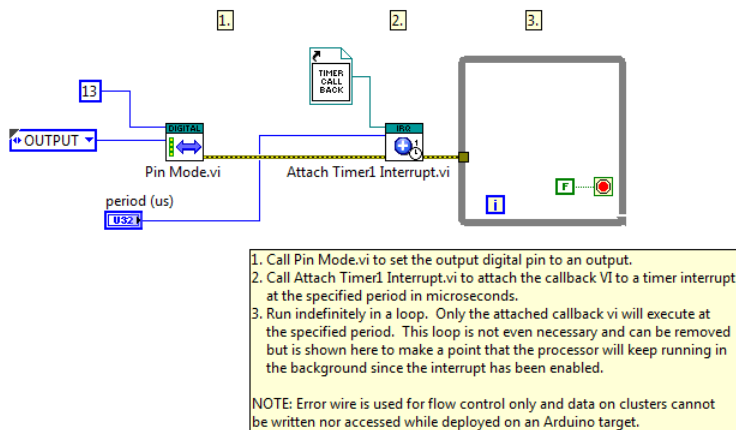
## Interrupt - Timer Example

Installed With: Arduino Compatible Compiler for LabVIEW

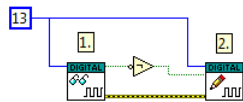
This example shows how to setup a microsecond based timer to trigger an interrupt to call into a callback VI. In the callback VI outputs I/O can be modified directly, or global variables can be used to update variables shared with the main VI. This example uses the onboard LED on pin 13.



## Main VI:



## Callback VI:

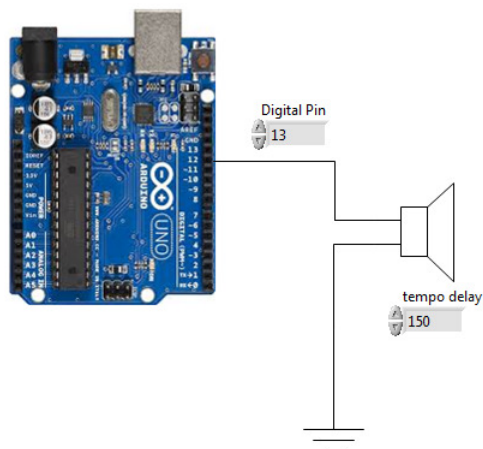


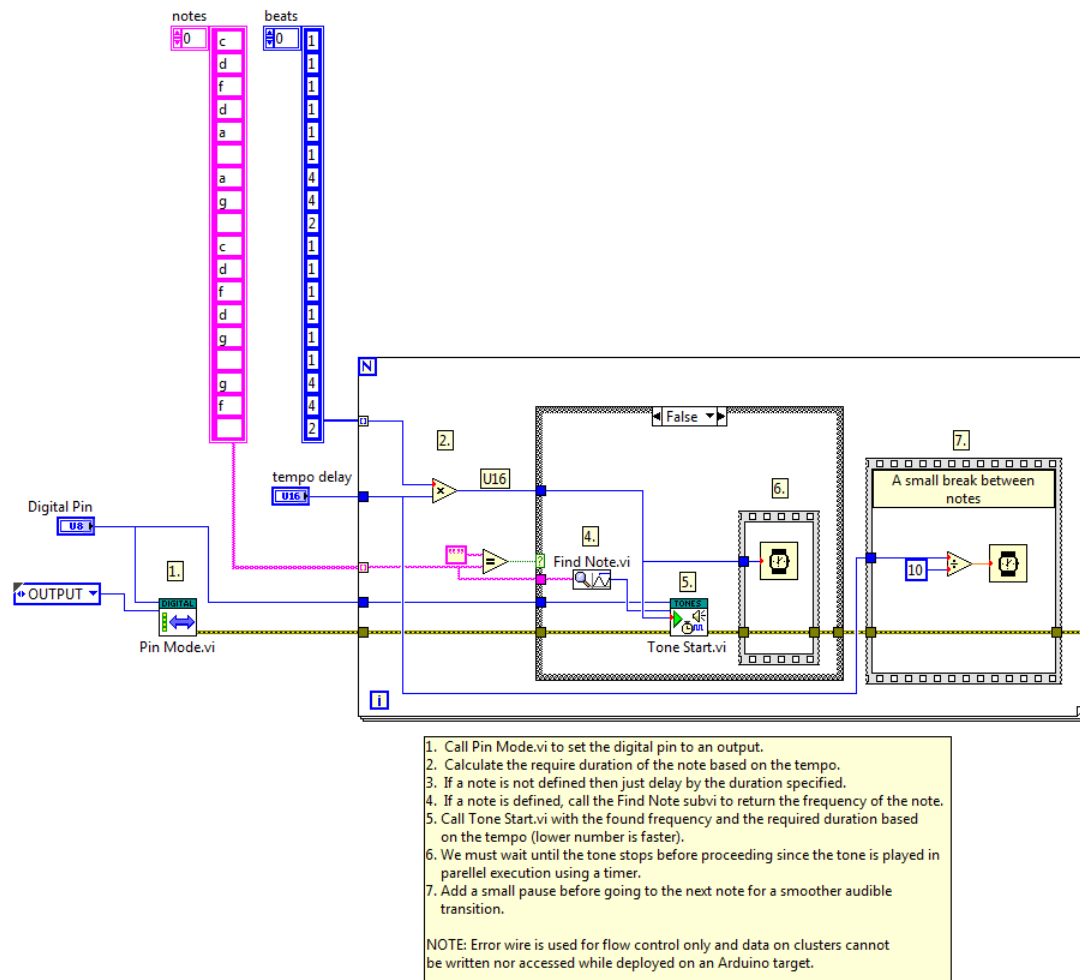
©2015 TSXperts/Aledyne. All rights reserved

### Tone - Play Song Example

Installed With: Arduino Compatible Compiler for LabVIEW

This example shows how to use the tone generator on a digital pin to play a song through a piezo buzzer. The tempo of the song can also be adjusted. The use of a subvi is also demonstrated as a lookup table to obtain the frequency of a specified tone. The tempo of the song can be changed by adjusting the control on the front panel (lower the number to make the song play faster) and re-download to observe the changes to the song.



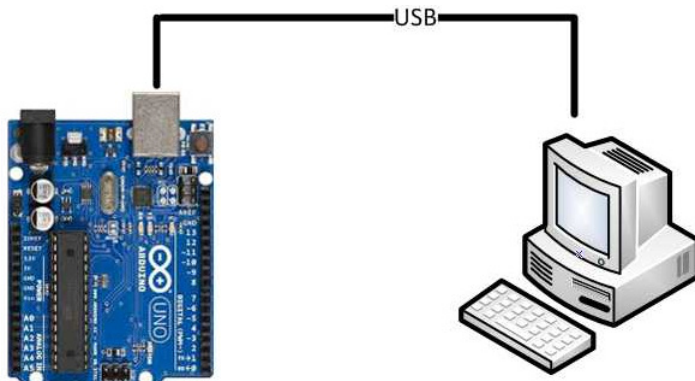


©2015 TSXperts/Aledyne. All rights reserved

### Serial - Monitoring GUI Example

Installed With: Arduino Compatible Compiler for LabVIEW

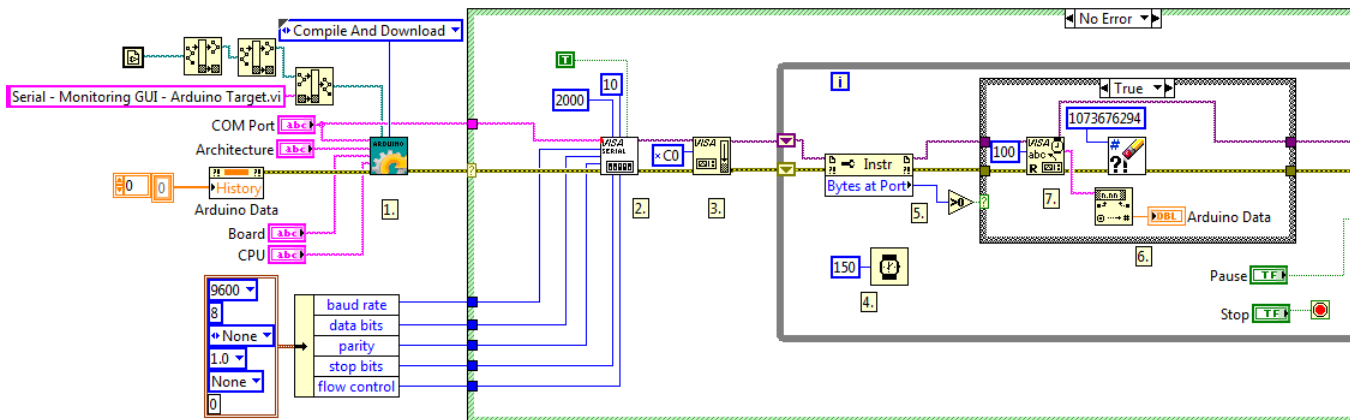
The Serial - Monitoring GUI example shows best practices for a remote monitoring system with two VIs, one running on a host PC and the other running embedded in the Arduino compatible target. The example includes two-way serial communication between the host PC VI and the Arduino compatible target VI. The host PC monitors and charts a specific variable of interest that is updated in the embedded Arduino target VI. The host PC VI also includes the ability to pause and resume the sending of data points by the Arduino compatible target VI. The target VI generates a point by point sine wave that is charted by the host PC VI.



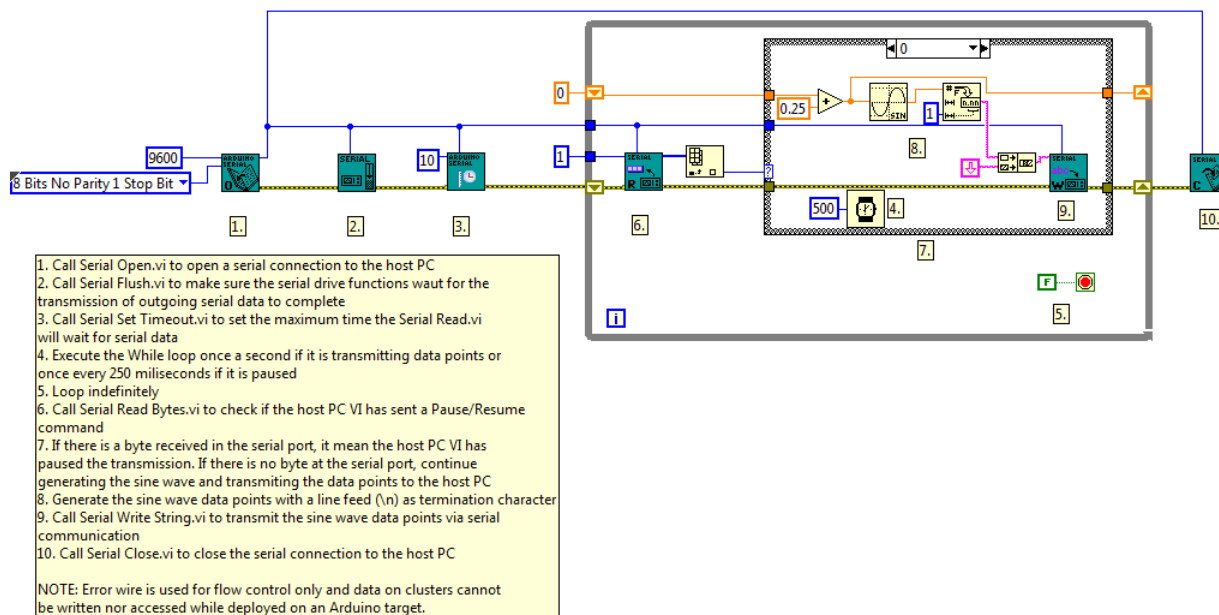
Host PC VI

1. Call Compiler.vi to compile and download to the Arduino target the Serial - Monitoring GUI - Arduino Target.vi.
2. Configure the serial communication parameters.
3. Flushes the serial buffer of any left over bytes.
4. Execute the monitoring code at every 150ms.
5. Check serial port for incoming bytes.
6. If there are bytes at the serial port, read them and display them in the Waveform Chart.
7. Read the incoming serial bytes, convert them to double data type and update the Waveform Chart.
8. If the Pause button is depressed, send a byte in the serial port, which will cause the Arduino target VI to pause sending data points via serial communication. If the Pause button is not depressed, it won't send the pause byte.
9. Transmit the pause byte via serial communication
10. Close the serial communication
11. Display any errors

NOTE: Error wire is used for flow control only and data on clusters cannot be written nor accessed while deployed on an Arduino target.



### Arduino Compatible Target VI

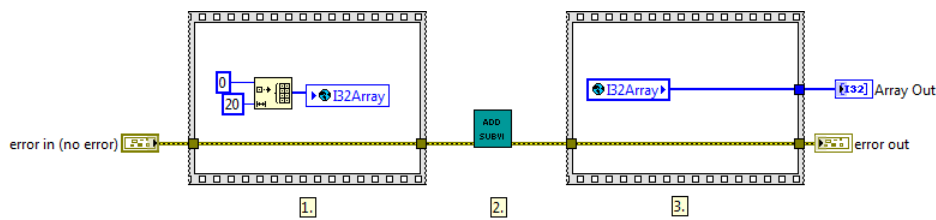


©2015 TSXperts/Aledyne. All rights reserved

### Memory Optimization - SubVIs Example

Installed With: Arduino Compatible Compiler for LabVIEW

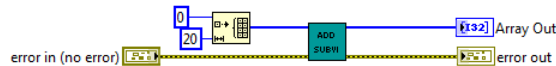
The Memory Optimization example demonstrates how to pass large data sets to subVIs using Global Variables. This is the preferred method for passing large data sets to subVIs as it avoids the creation of extra memory copies of those data sets inside the subVIs. Refer to the Important Considerations Before Start Creating Arduino VIs section of the Quick Start Guide for more information on Memory Optimization techniques.



1. Create an array of 20 elements and updates the Global Variable with the created array
2. Executes subVI that operates in place on the Global Variable without creating an extra memory copy for the array
3. Retrieves the updated array and passes it onto the Array Out indicator

The functionality of the code above is the equivalent of the code below, but in the code below, an extra memory copy of the array is created inside the subVI. Therefore, the code above is more memory efficient.

NOTE: Error wire is used for flow control only and data on clusters cannot be written nor accessed while deployed on an Arduino target.



## Saving Read-Only Data to Program Memory Example

**Installed With:** Arduino Compatible Compiler for LabVIEW

This example demonstrates how to store data in flash (program) memory instead of SRAM and read it back within a running program. This provides a way to tell the compiler to put a large array into flash memory instead of into SRAM, where it would normally go. The data stored in program memory is only available as read-only to your program. This is a good way to get use of more memory for things like lookup tables. You must use the Read Program Memory functions to retrieve the data from your program at run-time.

**Overview:** Demonstrates how to store data in flash (program) memory instead of SRAM and read it back within a running program. This provides a way to tell the compiler to put a large array into flash memory instead of into SRAM, where it would normally go. The data stored in program memory is only available as read-only to your program. This is a good way to get use of more memory for things like lookup tables. You must use the Read Program Memory functions to retrieve the data from your program at run-time.

NOTE: The input array must be a control or constant and must be directly wired to this function as well as the Name to assign to the array.

### Instructions:

1. If this example has been loaded from the Arduino Compatible Compiler for LabVIEW, simply select **Compile and Download**. Otherwise, open **Tools>Arduino Compatible Compiler for LabVIEW** in LabVIEW, then select **File>Load VI**, and select this VI and then **Compile and Download**.
2. Open the Serial Monitor and observe the data being written matches the data stored in the constant array below.



Notice in the compilation results that the program space is almost completely used because in addition to the code, the input 10,000 element U16 array (20,000 bytes) is being stored in program space. This accounts for 62% of available program space, and the additional 12% taken by the actual program. Therefore, it can be known at compile time if there is enough memory to fit both the code and the lookup table in memory.

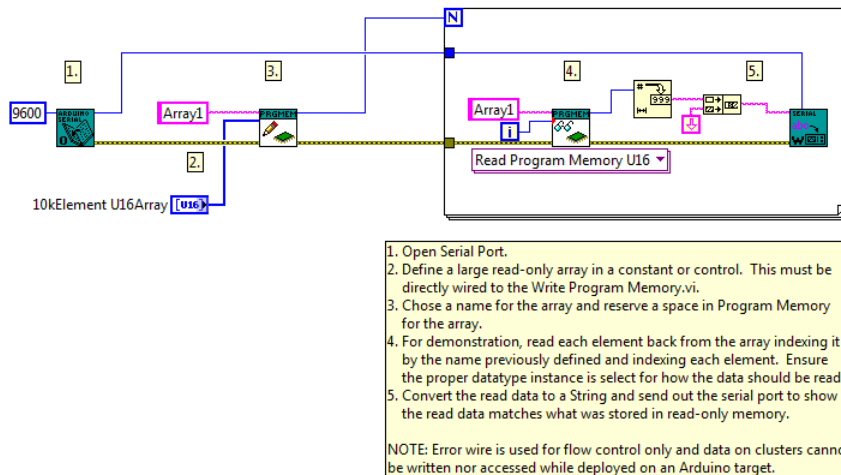
```
Compiling "C:\Projects\Arduino\LabVIEW4Arduino\examples\Memory Optimization\Saving
Read-Only Data to Program Memory.vi"

Running interpreter...Finished!
Generating downloadable program...Finished!
Compiling...Finished!

Loading configuration...
Initializing packages...
Preparing boards...
Verifying...

Sketch uses 24,074 bytes (74%) of program storage space. Maximum is 32,256 bytes.
Global variables use 212 bytes (10%) of dynamic memory, leaving 1,836 bytes for local
variables. Maximum is 2,048 bytes.
```



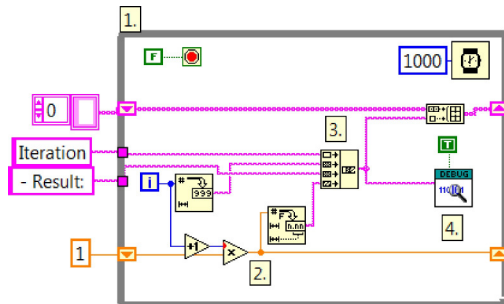


©2015 TSXperts/Aledyne. All rights reserved

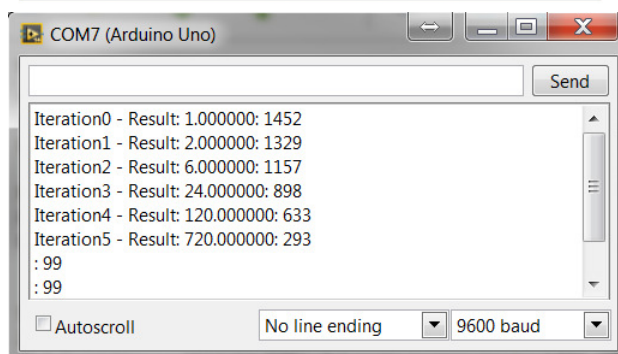
### Debugging - Memory Crash Example

**Installed With:** Arduino Compatible Compiler for LabVIEW

Demonstrates how to use the Debug Tool API VI in conjunction with the Arduino Compatible Compiler for LabVIEW Serial Monitor to debug an embedded Arduino VI. It also demonstrate how to look for memory leaks on an application that doesn't generate the expected results even though it doesn't generate a compilation error. Notice in the Serial Monitor how the available memory value drops per iteration until the output on the serial port stops being correct based on what the implemented code is doing.



1. Configure an infinite loop executing once a second.
  2. Calculation is converted to string and appended to data to be sent to serial port.
  3. Append string value that is built into the array along with calculation result to be sent to serial port by Debug Tool.
  4. Call Debug Tool.vi to send the formatted string to the serial port.
- Note that the Get Free Memory(F) input is set to True. This signals the Debug Tool.vi to grab the available free memory and append its number of bytes to the formatted string to be sent to the serial port.

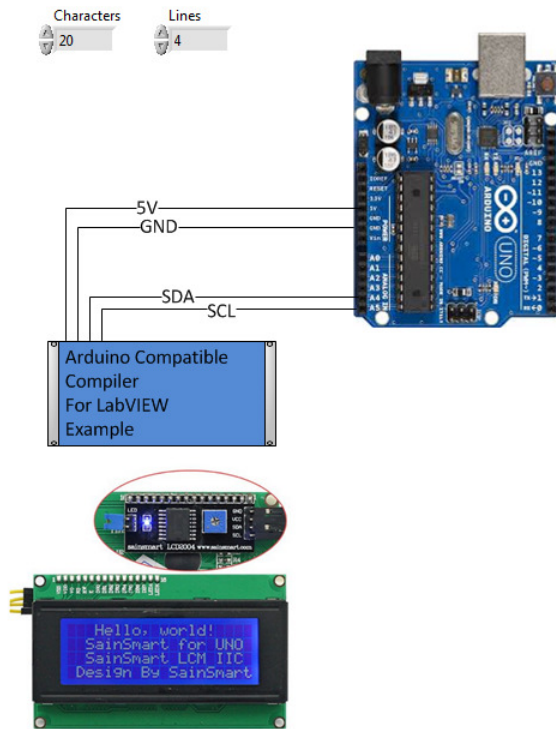


©2015 TSXperts/Aledyne. All rights reserved

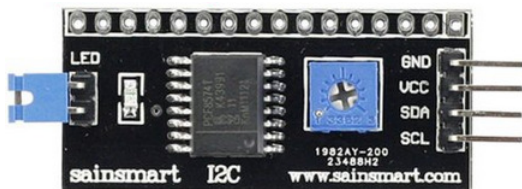
## I2C LCD - 4 Line LCD Example

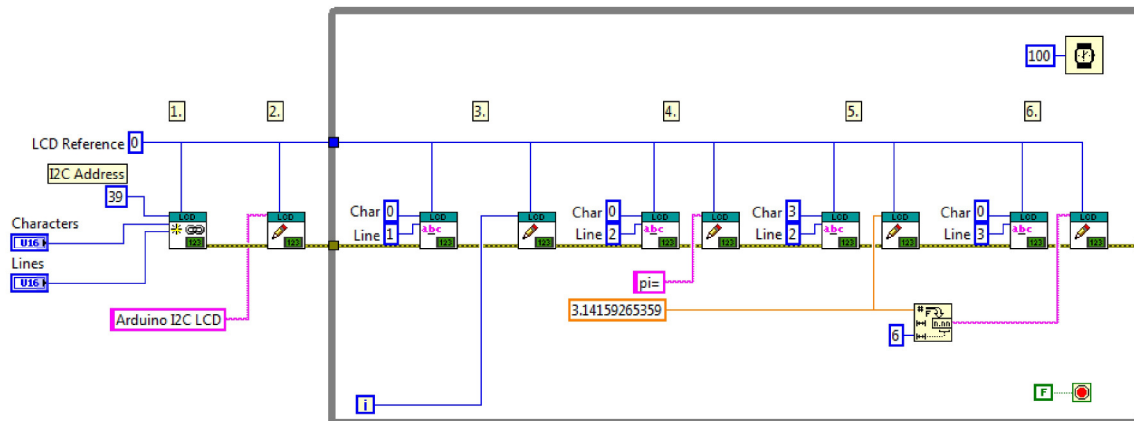
**Installed With:** Arduino Compatible Compiler for LabVIEW

This example demonstrates how to initialize and write data to an I2C based LCD screen. To use the I2C LCD library, most parallel LCDs can be used with a low cost SainSmart LCD I2C Adapter. The purpose of the adapter is to cut down the amount of conductors needed by using a serial I2C interface to the LCD. This example assumes a SainSmart I2C LCD or I2C Adapter is being used. This example has been tested with the following LCDs: SainSmart IIC/I2C/TWI Serial 2004 20x4 LCD Module Shield. The LiquidCrystal\_I2C Express.vi initialization is demonstrated here, which hardcodes the interface pins between the LCD module and the parallel LCD.



SainSmart I2C LCD Adapter can be used to convert the parallel interface of most LCD's to an I2C interface. Some SainSmart LCD's already have this converter built into the backside of the LCD, like the one pictured above, which is a 20x4 LCD Module Shield. These adapters are known to either use I2C Address 0x3F (63) or 0x27 (39). Make sure to change the address on the block diagram accordingly.





1. Call LiquidCrystal\_I2C Express.vi to initialize the LCD for the correct number of characters and lines and to configure the I2C address and pins on the SainSmart I2C LCD Adapter board. The backlight pin and homing is also handled in the Express V.I.
2. Call LiquidCrystal\_I2C write String.vi to write a string to the first line.
3. Set the cursor to the second line using LiquidCrystal\_I2C set cursor.vi and write the loop counter to the first character offset.
4. Set the cursor to the third line and write the "pi=" label to the first character offset.
5. Move the cursor over 3 characters (past the "pi=" label) and write the value of pi using a LiquidCrystal\_I2C write Double.vi. Notice that when writing single and double floating point numbers straight to the LCD, they will automatically be written with a precision of 2.
6. Convert the floating point number to a fractional string with a precision of 6 and now note that the precision of the string written is that specified in the conversion using LiquidCrystal\_I2C write String.vi.

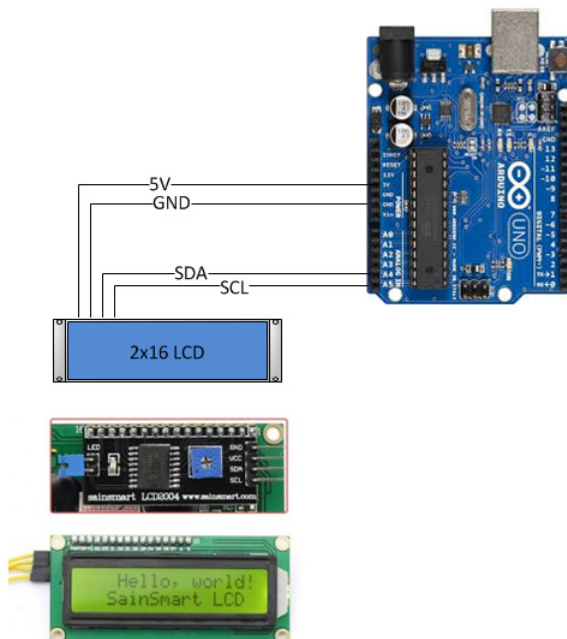
NOTE: Error wire is used for flow control only and data on clusters cannot be written nor accessed while deployed on an Arduino target.

©2015 TSXperts/Aledyne. All rights reserved

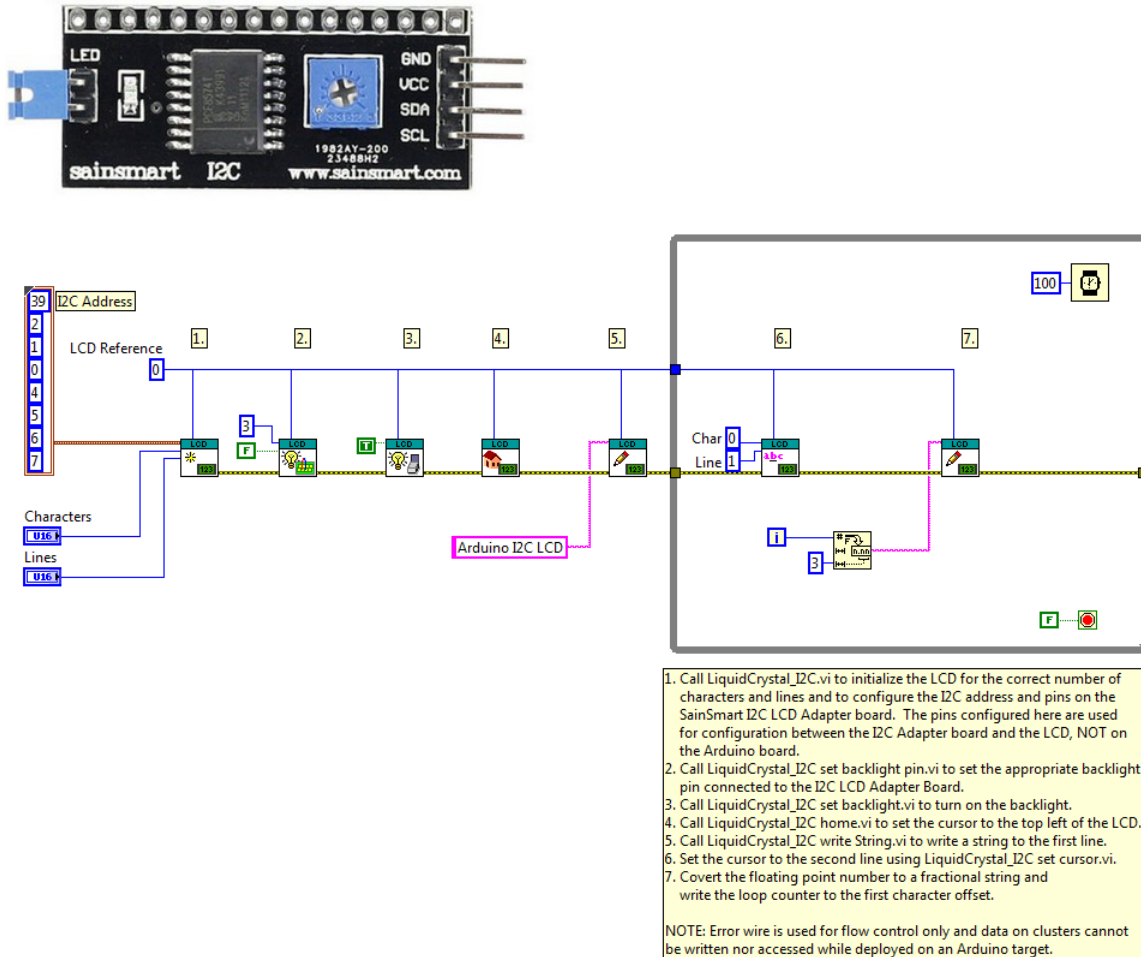
## I2C LCD - 2 Line LCD Example

**Installed With:** Arduino Compatible Compiler for LabVIEW

This example demonstrates how to initialize and write data to a 2 Line, 16 Character I2C based LCD screen. To use the I2C LCD library, most parallel LCDs can be used with a low cost SainSmart LCD I2C Adapter. The purpose of the adapter is to cut down the amount of conductors needed by using a serial I2C interface to the LCD. This example assumes a SainSmart I2C LCD or I2C Adapter is being used. This example has been tested with the following LCDs: SainSmart IIC/I2C/TWI Serial 2004 16x2 LCD Module Shield.



SainSmart I2C LCD Adapter can be used to convert the parallel interface of most LCD's to an I2C interface. Some SainSmart LCD's already have this converter built into the backside of the LCD, like the one pictured above, which is a 16x2 LCD Module Shield. These adapters are known to either use I2C Address 0x3F (63) or 0x27 (39). Make sure to change the address on the block diagram accordingly.

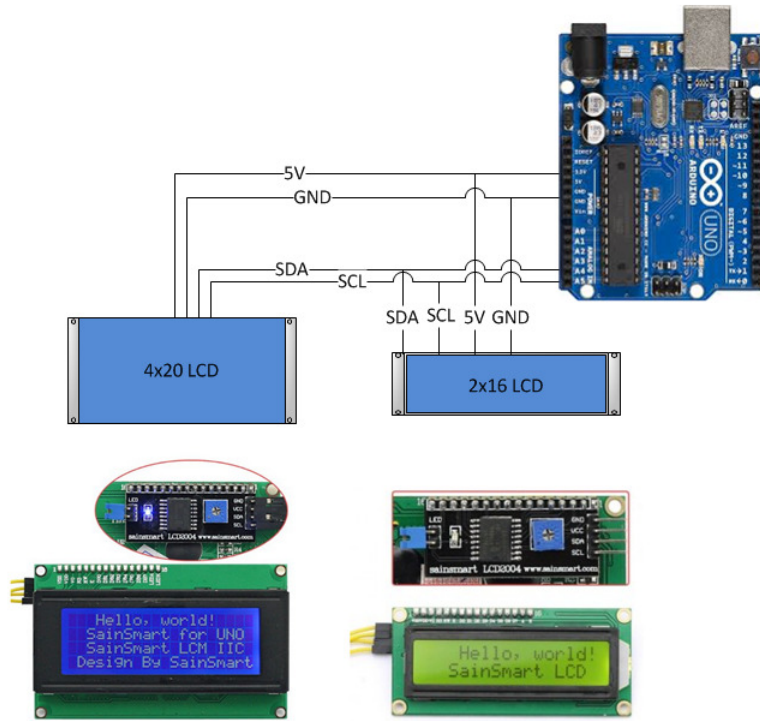


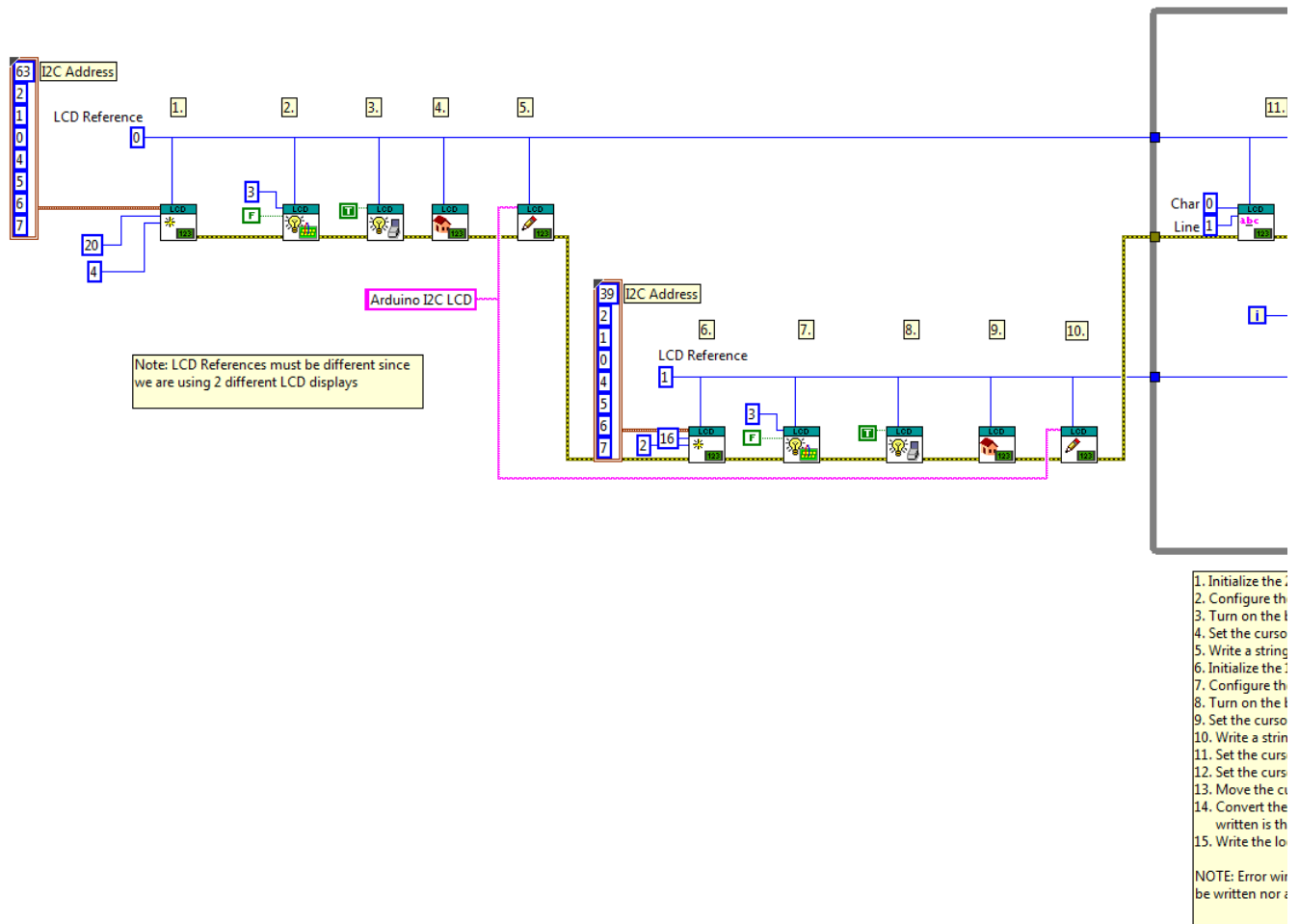
©2015 TSXperts/Aledyne. All rights reserved

### I2C LCD - Simultaneous LCD Control Example

Installed With: Arduino Compatible Compiler for LabVIEW

Demonstrates how to initialize and write data to multiple I2C LCD screens at the same time. Separate LCD instances are used to address each LCD in the same execution loop. Each LCD must have a different I2C address since they are wired to the same I2C bus. This example has been tested with the following LCDs: SainSmart IIC/I2C/TWI Serial 2004 20x4 LCD Module Shield and SainSmart IIC/I2C/TWI Serial 2004 16x2 LCD Module Shield. The 4x20 LCD uses I2C address 0x3F and the 2x16 LCD uses I2C Address 0x27.





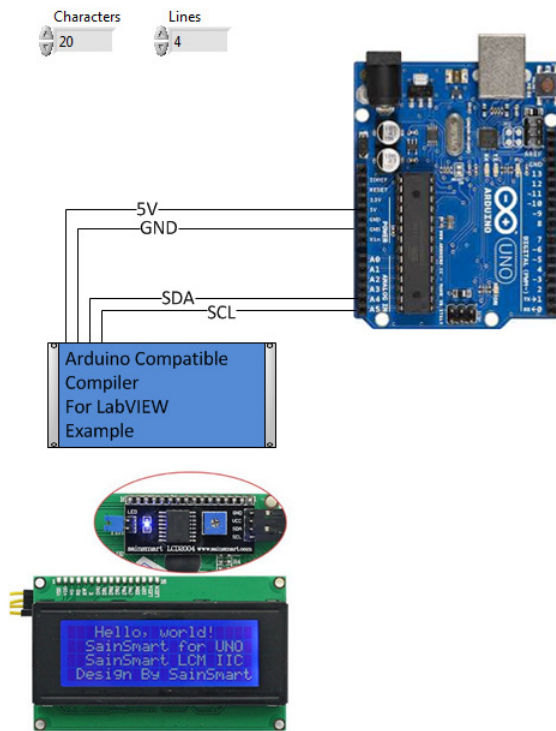
©2015 TSXperts/Aledyne. All rights reserved

### EEPROM Example

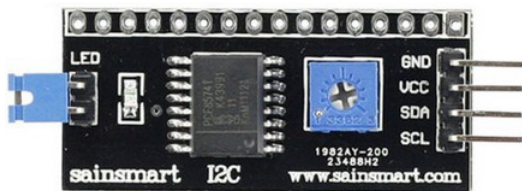
**Installed With:** Arduino Compatible Compiler for LabVIEW

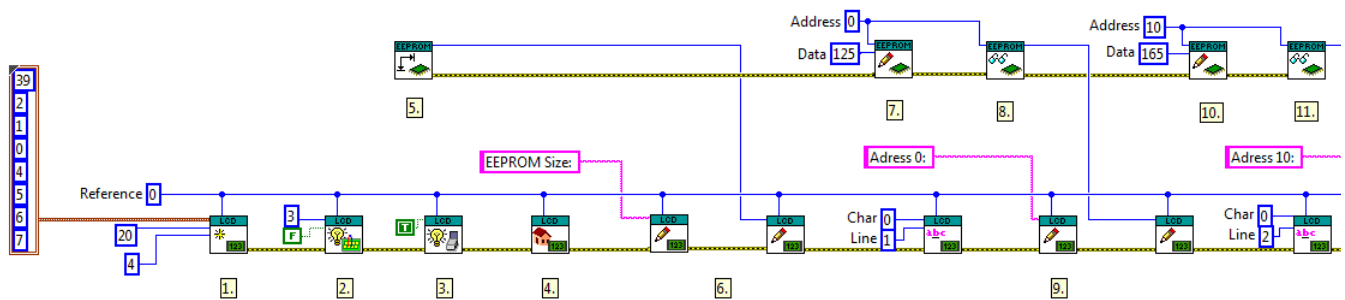
This example demonstrates how to read and write data to the Arduino EEPROM. Refer to the documentation of your specific Arduino target for more information on its EEPROM. This example retrieves the size of your Arduino EEPROM, writes values to two independent EEPROM addresses and read those two values back. The three data points are displayed on an I2C based LCD screen. Refer to the LCD shipping examples and product documentation for more information on how to use the I2C LCD library. This example assumes a Sainsmart I2C LCD or I2C Adapter is being used. This example has been tested with the following LCDs: SainSmart IIC/I2C/TWI Serial 2004 20x4 LCD Module Shield.





SainSmart I2C LCD Adapter can be used to convert the parallel interface of most LCD's to an I2C interface. Some SainSmart LCD's already have this converter built into the backside of the LCD, like the one pictured above, which is a 20x4 LCD Module Shield. These adapters are known to either use I2C Address 0x3F (63) or 0x27 (39). Make sure to change the address on the block diagram accordingly.





1. Call LiquidCrystal\_I2C.vi to initialize the LCD for the correct number of characters and lines and to configure the I2C address and pins on the SainSmart I2C LCD Adapter board. The pins configured here are used for configuration between the I2C Adapter board and the LCD, NOT on the Arduino board.
2. Call LiquidCrystal\_I2C set backlight pin.vi to set the appropriate backlight pin connected to the I2C LCD Adapter Board.
3. Call LiquidCrystal\_I2C set backlight.vi to turn on the backlight.
4. Call LiquidCrystal\_I2C home.vi to set the cursor to the top left of the LCD.
5. Call Get EEPROM size.vi to retrieve the size of the Arduino target EEPROM.
6. Call LiquidCrystal\_I2C write String.vi two times to write a string and the EEPROM size value returned from the Get EEPROM size.vi to the first line.
7. Call EEPROM Write.vi to write the value 125 to EEPROM address 0.
8. Call EEPROM Read.vi to read the value stored at EEPROM address 0.
9. Set the LCD cursor to the second line using LiquidCrystal\_I2C set cursor.vi, write a string and the value stored at EEPROM address 0.
10. Call EEPROM Write.vi to write the value 165 to EEPROM address 10.
11. Call EEPROM Read.vi to read the value stored at EEPROM address 10.
12. Set the cursor to the third line using LiquidCrystal\_I2C set cursor.vi, write a string and the value stored at EEPROM address 10.

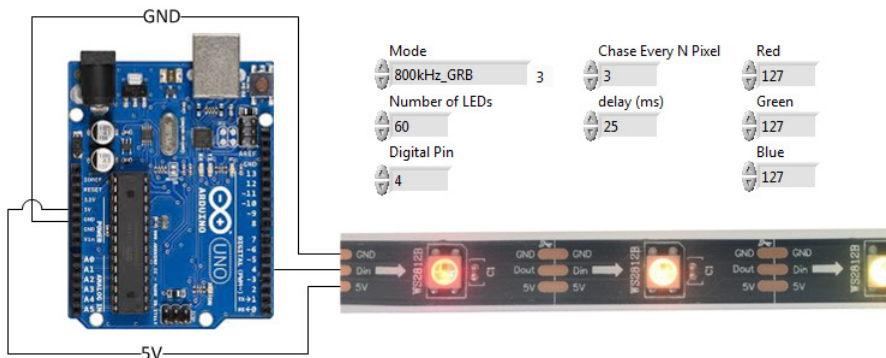
NOTE: Error wire is used for flow control only and data on clusters cannot be written nor accessed while deployed on an Arduino target.

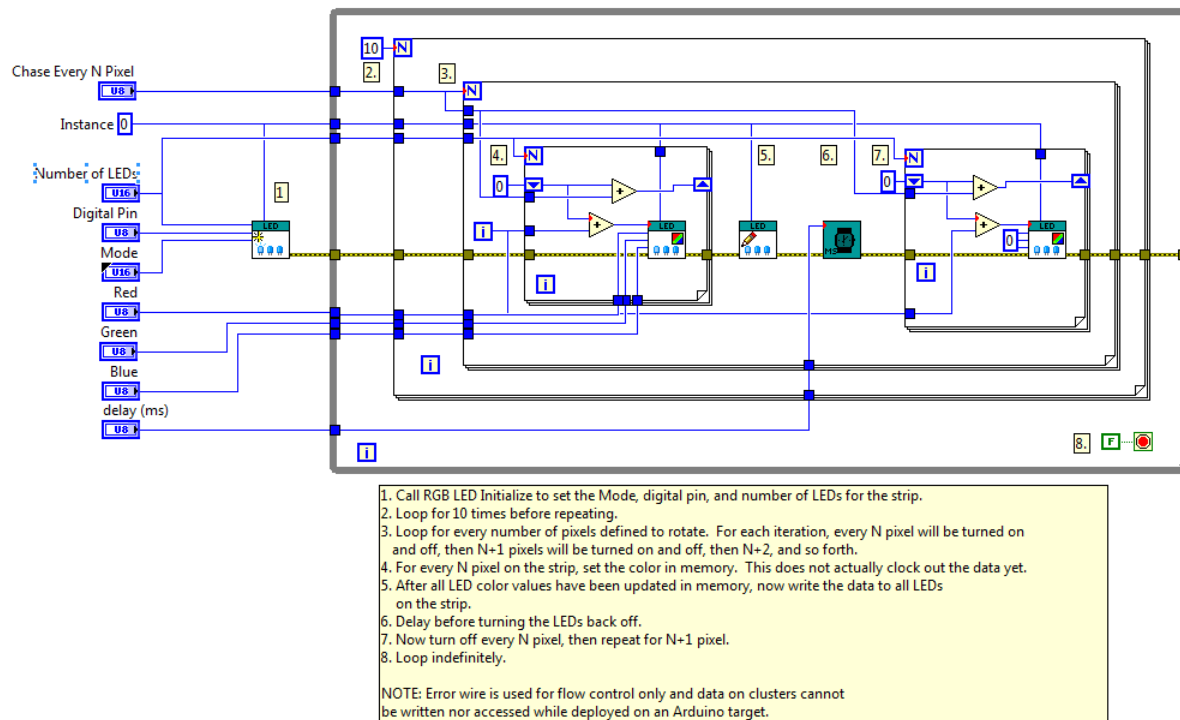
©2015 TSXperts/Aledyne. All rights reserved

### RGB LED - Chaser Example

Installed With: Arduino Compatible Compiler for LabVIEW

This example demonstrates a theater light chaser on a 1-wire RGB LED strip using a WS2811 or WS2812 controller. The color of all LEDs can be preset as well as the number of lights to cycle in the light chase. The time delay can also be adjusted to change the dramatic effect.



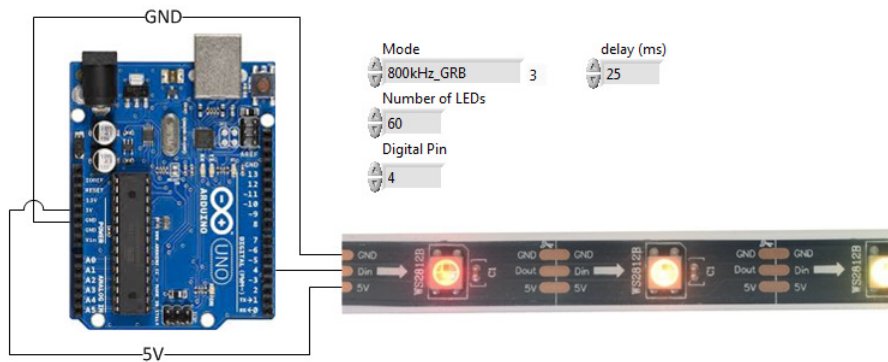


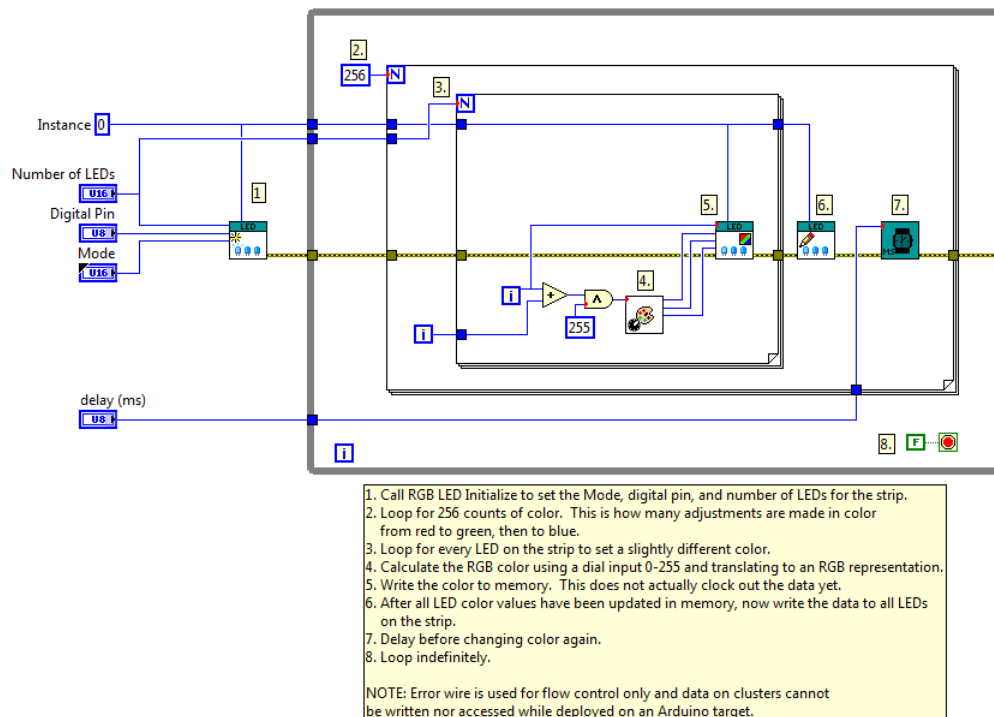
©2015 TSXperts/Aledyne. All rights reserved

### RGB LED - Rainbow Example

Installed With: Arduino Compatible Compiler for LabVIEW

This example demonstrates a rainbow effect on a 1-wire RGB LED strip using a WS2811 or WS2812 controller. Each LED in the strip is gradually changed from red to green, then to blue with a blend of colors in between. The LED color changing effect is staggered down the strip so not all LEDs are changed at the same time. The time delay can also be adjusted to change how fast the color is changed.



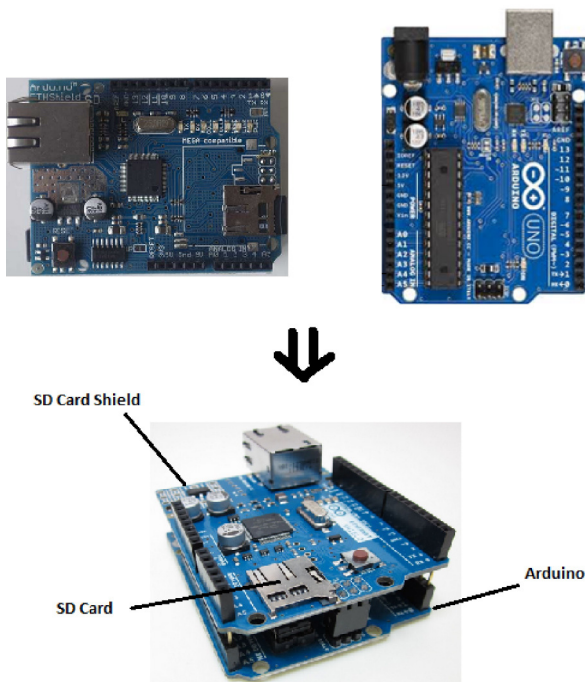


©2015 TSXperts/Aledyne. All rights reserved

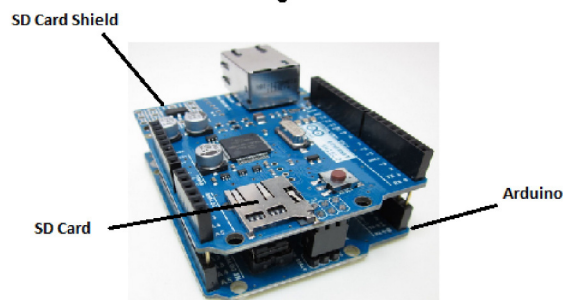
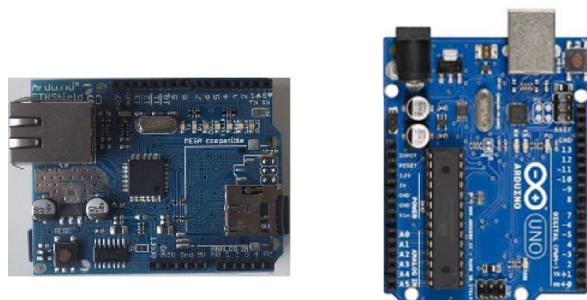
### SD Card - Read Data GUI Example

Installed With: Arduino Compatible Compiler for LabVIEW

The SD Card - Read Data example is to be executed in conjunction with the SD Card - Log Data shipping example. The project shows how to read data previously written to a SD Card and to transfer this data via serial communication. The project include two VIs, one running on a host PC and the other running embedded in the Arduino compatible target. Refer to your specific SD Card Shield documentation for its Chip Select (CS) pin number.

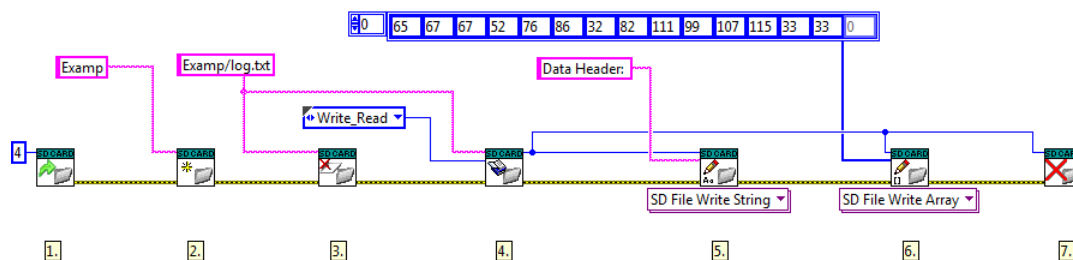






## Arduino Compatible Target VI

Note: SD Lib uses 8.3 filenames which have at most 8 characters optionally followed by a period . and a filename extension of at most three characters. File and directory names are uppercase, although systems that use the 8.3 standard are usually case-insensitive



1. Call SD Card Open.vi to iopen a reference to the SD Card.
2. Call SD Create Directory.vi to create a directory named Examp in the SD Card file system.
3. Call SD Remove File.vi to remove any previous instance of file named log.txt.
4. Call SD File Open.vi to open a reference to the file named log.txt. All subsequent operations to this file will require this reference as an input.
5. Call SD File Write String.vi to demonstrate the logging of a string to the log.txt file.
6. Call SD File Write Array.vi to demonstrate the logging of an array of bytes to the log.txt file. The Array of Bytes constant contains the ASCII representation of the sentence: "ACC4LV Rocks!!"
7. Call SD File Close.vi to close the reference to the log.txt file.

NOTE: Error wire is used for flow control only and data on clusters cannot be written nor accessed while deployed on an Arduino target.

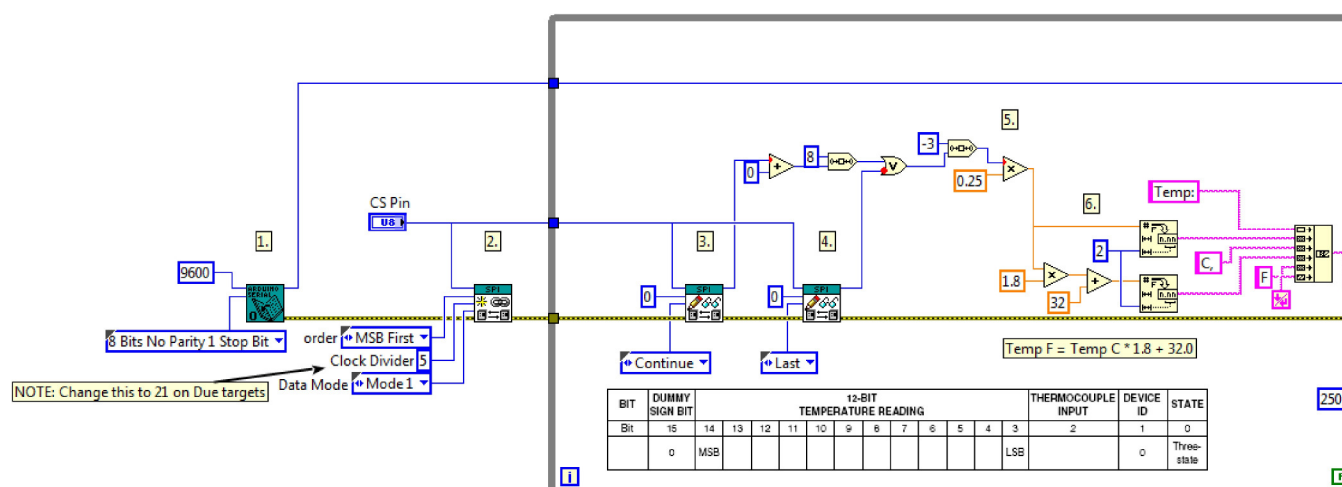
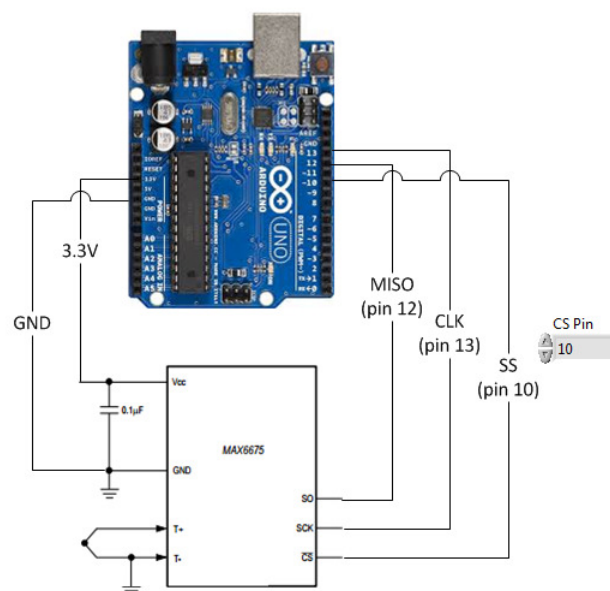
©2015 TSXperts/Aledyne. All rights reserved

## SPI - MAX6675 Thermocouple Example

Installed With: Arduino Compatible Compiler for LabVIEW

This example demonstrates how to read the temperature from a MAX6675 SPI K-Type Thermocouple to Digital converter. The temperature is read in Celsius, then converted to Fahrenheit and written to the serial port every 200ms. On AVR targets only one Chip Select pin (pin 10) is supported by the SPI peripheral, however on Due targets, other CS Pins can be selected on the front panel before downloading.





1. Call Serial Open.vi to open a serial port to write acquired data to.
2. Call SPI Open Express.vi to initialize the SPI peripheral on the selected CS pin. Also sets the CS pin as an output and raises it to initialize it, sets the Bit order of the SPI byte transactions, sets the Data Mode, and sets the Clock Divider. For AVR targets, 0 sets divide by 8, or 2MHz for 16MHz boards. For Due targets set this to 42 to achieve the same frequency (84MHz/42 = 2MHz).
3. Read one byte from the SPI bus. For the MAX6675 this is the high byte. Make sure to keep CS low for the 2 byte transaction so we set transfer mode to "Continue".
4. Now read the low byte and set transfer mode to Last so CS is raised after the transaction.
5. Convert the 2 bytes to a single uint. Note that to convert the high byte to a uint first, an add primitive is used with a unit constant of 0 as input to the other terminal to force a cast from the read high byte to a uint. Then the high byte is shifted left by 8 bits and Or'ed with the low byte. Then the result is shifted right by 3 bits and multiplied by 0.25 per the MAX6675 datasheet (Bits D14-D3 contain the converted temperature in the order of MSB to LSB). 1 bit is 0.25 degree so we must also divide by 4.
6. Convert Celsius to Fahrenheit and convert both to String datatypes.
7. Format a message containing both Celsius and Fahrenheit temperatures and send over the serial interface.
8. Loop indefinitely. Wait 250ms between reads as the MAX6675 maximum conversion time is 0.22s.

NOTE: Error wire is used for flow control only and data on clusters cannot be written nor accessed while deployed on an Arduino target.

## I2C - DS1307 Real Time Clock Example

Installed With: Arduino Compatible Compiler for LabVIEW

This example demonstrates how to set and read the current time from a DS1307 Real-Time Clock (RTC). This RTC is used on the Arduino

Datalogger Shield, which also comes equipped with an SD card slot. To set the current time, enable the "Set" control and populate the date structure with the current date/time, then download the code first. Then turn off "Set" and download again to read the time only. The current date/time will be maintained after power cycle if the battery is installed in the shield.

## DS1307 Register Table:

This is the table showing the register layout for the date/time stored in the DS1307. This example only shows writing and reading of register addresses 0-7.

ADDRESS	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0	FUNCTION	RANGE
00h	CH	10 Seconds			Seconds				Seconds	00–59
01h	0	10 Minutes			Minutes				Minutes	00–59
02h	0	12	10 Hour	10 Hour	Hours				Hours	1–12 +AM/PM 00–23
		24	PM/AM							
03h	0	0	0	0	0	DAY			Day	01–07
04h	0	0	10 Date			Date			Date	01–31
05h	0	0	0	10 Month	Month				Month	01–12
06h	10 Year			Year				Year	00–99	
07h	OUT	0	0	SQWE	0	0	RS1	RS0	Control	—
08h–3Fh									RAM 56 x 8	00h–FFh

0 = Always reads back as 0.

## DS1307 Real Time Clock VI:

### Datalogger Shield



+



Set

Date

0

seconds (0-59)

45

minutes (0-59)

12

hour (0-23)

0

Day (unused)

10

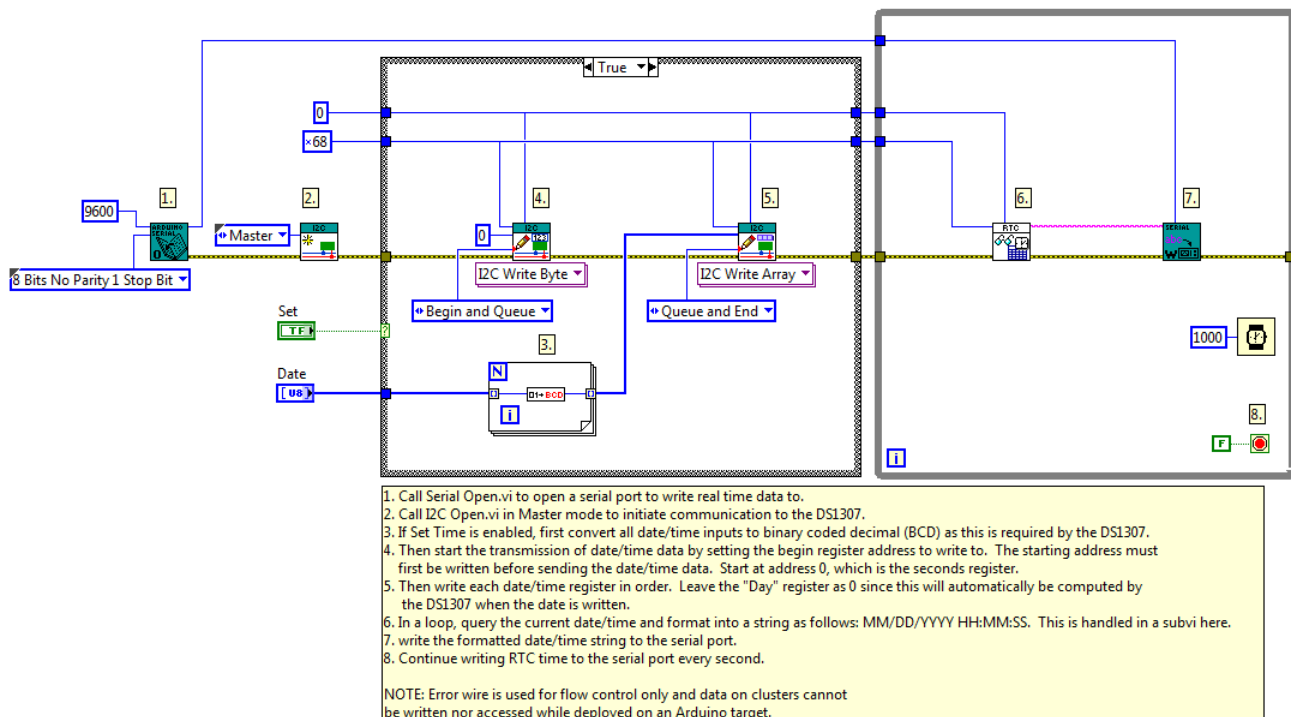
Date (1-31)

4

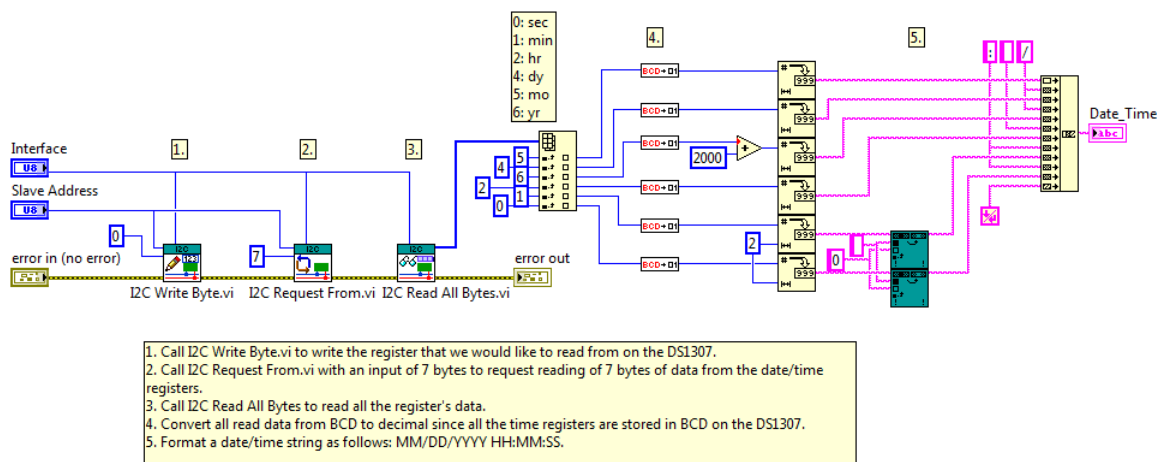
Month (1-12)

15

Year (0-99)



## Read RTC Time and Format VI:



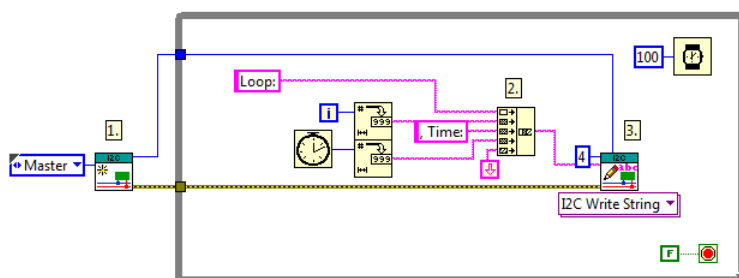
©2015 TSXperts/Aledyne. All rights reserved

### I2C - Master Slave Example

Installed With: Arduino Compatible Compiler for LabVIEW

This example demonstrates how to send data between 2 Arduino boards using the I2C interface. The Master (transmitting) end of the communication writes its loop and timing information to the Slave as a String. The Slave (receiving) end of the communication registers a callback VI to be run whenever an I2C receive interrupt occurs making this an interrupt driven message handler. When an I2C message comes in, the data is formatted and written to the serial terminal for demo purposes only.

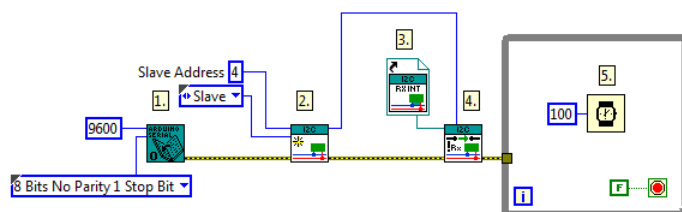
## I2C Master VI:



1. Call I2C Open.vi to open the I2C port as a Master.
2. Format a string message to be sent to the slave which includes the loop number and the millisecond tick count.
3. Send the message to the Slave device at I2C Address 4. This address must match the slave address set in the Slave Arduino (refer to I2C Slave.vi).

NOTE: Error wire is used for flow control only and data on clusters cannot be written nor accessed while deployed on an Arduino target.

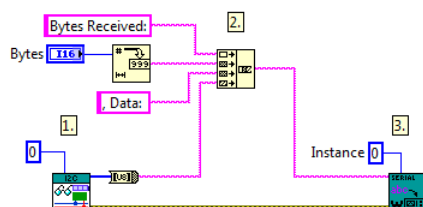
## I2C Slave VI:



1. Call Serial Open.vi to open a serial port to write data received from the master device. Note that the actual writing of data is done in the I2C receive interrupt callback but the port must be opened once during initialization. To address the correct port in the interrupt without using the Instance number from Serial Open.vi, use a hardcoded index correlating to the serial port number (refer to I2C Receive Interrupt.vi).
2. Call I2C Open.vi to open the I2C port as a Slave. Use slave address 4. This must match the slave address set in the master (refer to I2C Master.vi).
3. Callback reference to the I2C Receive Interrupt handler. This Callback will be called whenever data is sent from the master to this slave device.
4. Attached the Callback VI to the I2C receive interrupt.
5. At this point the standard loop does nothing and the interrupt handler received and processes the data. Any code that takes significant processor time, like LCD display control, would be placed in the main while loop and data can be shared between the callback and the main loop using global variables.

NOTE: Error wire is used for flow control only and data on clusters cannot be written nor accessed while deployed on an Arduino target.

## I2C Slave Receive Interrupt VI:

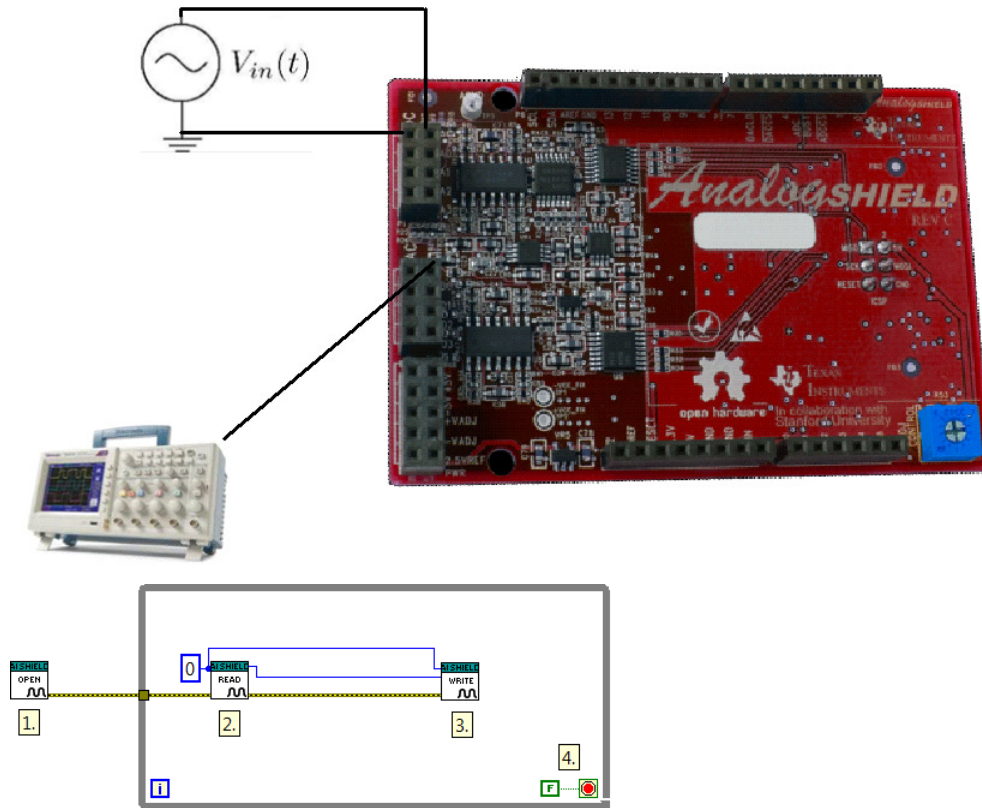


1. Call I2C Read All Bytes.vi to read all bytes from the I2C port.
2. Format the serial data to print for debugging including the number of bytes received from the interrupt and the data actually received. Note that the number of bytes received is automatically provided by the interrupt handler and this input must be wired to the connector pane of the callback VI for the I2C receive interrupt.
3. Write the serial message to the serial terminal. Note that to address Serial port 0, a hardcoded 0 can be used for the instance. If this is being used on a board with more than one serial port and it is desired to use a different port, the index can be used which correlates to that port (e.g. use '2' for port 2).

**Installed With:** Arduino Compatible Compiler for LabVIEW

This example demonstrates how to utilize the Digilent Analog Shield <https://www.digilentinc.com/Products/Detail.cfm?NavPath=2,648,1261&Prod=TI-ANALOG-SHIELD>

to acquire analog data from a data source connected to an input analog pin on the shield and output the acquired data to an analog output pin on the shield. The analog output data will be exactly the same as the acquired analog input data.



1. Call Open.vi to open a reference to the Analog Shield.
2. Call Read.vi to read the voltage on the analog pin specified.
3. Call the Write.vi to output the analog value read to the analog channel specified.
4. Loop indefinitely.

NOTE: Error wire is used for flow control only and data on clusters cannot be written nor accessed while deployed on an Arduino target.